UNCLASSIFIED

| AD NUMBER |
| --- |
| **ADB129569** |
| LIMITATION CHANGES |
| TO:<br>**Approved for public release; distribution is unlimited.** |
| FROM:<br>**Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; MAR 1988. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL 32542. This document contains export-controlled technical data.** |
| AUTHORITY |
| **AFSC ltr dtd 13 Feb 1992** |

THIS PAGE IS UNCLASSIFIED

DTIC FILE COPY

(2)

AFATL-TR-88-62, VOL II

# Common Ada Missile Packages-Phase 2 (CAMP-2)

## Volume II. 11th Missile Demonstration

AD-B129 569

D McNicholl
C Palmer
J Mason, et al.

McDONNELL DOUGLAS ASTRONAUTICS COMPANY
P O BOX 516
ST LOUIS, MO 63166

DTIC
ELECTE
DEC 1 2 1988
S D
E

NOVEMBER 1988

FINAL REPORT FOR PERIOD SEPTEMBER 1985-MARCH 1988

CRITICAL TECHNOLOGY

Distribution authorized to U.S. Government agencies and their contractors only; ~~this report documents test and evaluation~~; distribution limitation applied March 1988. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG) Eglin Air Force Base, Florida 32542-5434.

DESTRUCTION NOTICE - For classified documents, follow the procedures in DoD 5220.22-M, Industrial Security Manual, Section II-19 or DoD 5200.1-R, Information Security Program Regulation, Chapter IX. For unclassified, limited documents, destroy by any method that will prevent disclosure of contents or reconstruction of the document.

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

88 12

# NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise as in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

Stephen C Korn

STEPHEN C. KORN
Chief, Aeromechanics Division

Even though this report may contain special release rights held by the controlling office, please do not request copies from the Air Force Armament Laboratory. If you qualify as a recipient, release approval will be obtained from the originating activity by DTIC. Address your request for additional copies to:

Defense Technical Information Center
Cameron Station
Alexandria, VA 22304-6145

If your address has changed, if you wish to be removed from our mailing list, or if your organization no longer employs the addressee, please notify AFATL/FXG, Eglin AFB, FL 32542-5434, to help us maintain a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

# REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

| | |
|---|---|
| 1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED | 1b. RESTRICTIVE MARKINGS: CRITICAL TECHNOLOGY |
| 2a. SECURITY CLASSIFICATION AUTHORITY | 3. DISTRIBUTION/AVAILABILITY OF REPORT: Distribution authorized to U.S. Government Agencies and their contractors; ~~this report contains test and evaluation~~ (OVER) |
| 2b. DECLASSIFICATION/DOWNGRADING SCHEDULE | |
| 4. PERFORMING ORGANIZATION REPORT NUMBER(S) | 5. MONITORING ORGANIZATION REPORT NUMBER(S): AFATL-TR-88-62, Volume II |
| 6a. NAME OF PERFORMING ORGANIZATION: McDonnell Douglas Astronautics Company; 6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION: Aeromechanics Division, Guidance and Control Branch |
| 6c. ADDRESS (City, State, and ZIP Code): P.O. Box 516, St Louis MO 63166 | 7b. ADDRESS (City, State, and ZIP Code): Air Force Armament Laboratory, Eglin Air Force Base, Florida 32542-5434 |
| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION: STARS Joint Program Office; 8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F08635-86-C-0025 |
| 8c. ADDRESS (City, State, and ZIP Code): Room 3D139 (1211 Fern St), The Pentagon, Washington DC 20301-3081 | 10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. 63756D; PROJECT NO. 921D; TASK NO GT; WORK UNIT ACCESSION NO. 02 |

11. TITLE (Include Security Classification) Common Ada Missile Packages-Phase 2 (CAMP-2), Volume II: 11th Missile Demonstration

12. PERSONAL AUTHOR(S)
D.G. McNicholl, J.F. Mason, C. Palmer, T.T. Taylor, and L.A. Finch

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, Month, Day) | 15. PAGE COUNT |
|---|---|---|---|
| Final | FROM Sep 85 TO Mar 88 | November 1988 | 128 |

16. SUPPLEMENTARY NOTATION

Availability of this report is specified on verso of front cover. (OVER)

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | Reusable Software, Missile Software, Software Generators, Ada parts, Composition, Systems, Software Parts |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)


The CAMP project, primarily funded ty the STARS Joint Program Office, sponsored by the Air Force Armament Laboratory, and performed by McDonnell Douglas, has taken a pragmatic approach to demonstrating the feasibility and utility of the concept of software reuse for real-time embedded missile systems. CAMP products include: 452 operational flight software parts in Ada for tactical missiles, and a prototype parts engineering system to support parts identification, cataloging and construction. In order to demonstrate the value of the reuse concept, a missile subsystem was built using the CAMP parts. Results indicate a significant increase in software productivity when developing systems using parts, Ada, modern software engineering practice, robust software tools, and knowledgeable software engineers.

This report is documented in three volumes: Volume I - CAMP Parts and Parts Composition System, Volume II - 11th Missile Demonstration, and Volume III - CAMP Armonics Benchmarks.


| | |
|---|---|
| 20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS | 21. ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED |
| 22a. NAME OF RESPONSIBLE INDIVIDUAL: Christine M. Anderson | 22b. TELEPHONE (Include Area Code): (904) 882-2961; 22c. OFFICE SYMBOL: AFATL/FXG |

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

distribution limitation applied March 1988. Other requests for this document must be referred to the Air Force Armament Laboratory (FXG), Eglin Air Force Base, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

## TRADEMARKS

The following table lists the trademarks used throughout this document:

| TRADEMARK | TRADEMARK OF |
|---|---|
| ACT | Advanced Computer Techniques |
| ART | Inference Corporation |
| ART Studio | Inference Corporation |
| CMS | Digital Equipment Corporation |
| DEC | Digital Equipment Corporation |
| Mikros | Mikros, Inc. |
| Oracle | Oracle Corporation |
| Scribe | Scribe Systems |
| Symbolics | Symbolics, Inc. |
| Symbolics 3620 | Symbolics, Inc. |
| TLD | TLD Systems Ltd |
| VAX | Digital Equipment Corporation |
| VMS | Digital Equipment Corporation |

# PREFACE

This report describes the work performed, the results obtained, and the conclusions reached during the Common Ada Missile Packages Phase-2 (CAMP-2) contract (F08635-86-C-0025). This work was performed by the Software and Information Systems Department of the McDonnell Douglas Astronautics Company, St. Louis, Missouri (MDAC-STL), and was sponsored by the United States Air Force Armament Laboratory (FXG) at Eglin Air Force Base, Florida. This contract was performed between September 1985, and March 1988.

The MDAC-STL CAMP program manager was:


Dr. Daniel G. McNicholl
Technology Branch
Software and Information Systems Department
McDonnell Douglas Astronautics Company
P.O. Box 516
St. Louis, Missouri 63166


The AFATL CAMP program manager was:


Christine M. Anderson
Guidance and Control Branch
Aeromechanics Division
Air Force Armament Laboratory
Eglin Air Force Base, Florida 32542-5434


This report consists of three volumes. Volume I contains information on the development of the CAMP parts and the Parts Composition System. Volume II contains the results of the 11th Missile Application development. Volume III contains the results of the CAMP Armonics Benchmarks Suite development.

Commercial hardware and software products mentioned in this report are sometimes identified by manufacturer or brand name. Such mention is necessary for an understanding of the R & D effort, but does not constitute endorsement of these items by the U.S. Government.

# ACKNOWLEDGEMENT


Special thanks to the Armament Division Deputy for Armament Control Office; to the Software Technology for Adaptable, Reliable Systems (STARS) Joint Program Office; to the Ada Joint Program Office (AJPO); and to the Air Force Electronic Systems Division, Computer Resource Management Technology Program Office for their support of this project.



DTIC COPY INSPECTED 6

| Accession For | |
|---|---|
| NTIS GRA&I | ☐ |
| DTIC TAB | ☒ |
| Unannounced | ☐ |
| Justification | |
| By | |
| Distribution/ | |
| Availability Codes | |
| Dist | Avail and/or Special |
| C-2 | |

# TRADEMARKS

The following table lists the trademarks used throughout this document:

| TRADEMARK | TRADEMARK OF |
|---|---|
| ACT | Advanced Computer Techniques |
| ART | Inference Corporation |
| ART Studio | Inference Corporation |
| CMS | Digital Equipment Corporation |
| DEC | Digital Equipment Corporation |
| Mikros | Mikros, Inc. |
| Oracle | Oracle Corporation |
| Scribe | Scribe Systems |
| Symbolics | Symbolics, Inc. |
| Symbolics 3620 | Symbolics, Inc. |
| TLD | TLD Systems Ltd |
| VAX | Digital Equipment Corporation |
| VMS | Digital Equipment Corporation |

# Table of Contents

## Table of Contents (cont'd)

## Table of Contents (cont'd)

## List of Figures

## List of Figures (cont'd)

# List of Tables

## List of Acronyms

| | |
|---|---|
| ACS | Ada Compilation System |
| ACVC | Ada Compiler Validation Capability |
| AdaJUG | Ada/Jovial Users Group |
| ADL | Ada Design Language |
| AFATL | Air Force Armament Laboratory |
| AFB | Air Force Base |
| AI | Artificial Intelligence |
| AJPO | Ada Joint Program Office |
| AMPEE | Ada Missile Parts Engineering Expert (System) |
| AMRAAM | Advanced Medium Range Air-to-Air Missile |
| ANSI | American National Standards Institude |
| APSE | Ada Programming Support Environment |
| Armonics | Armament Electronics |
| ART | Automated Reasoning Tool |
| ASCII | American Standard Code for Information Interchange |
| BC | Bus Controller |
| BDT | Basic Data Types |
| BIM | Bus Interface Module |
| CAD/CAM | Computer-Aid Design/Computer-Aided Manufacturing |
| CAMP | Common Ada Missile Packages |
| CCCB | Configuration Change Control Board |
| CDRL | Contractual Data Requirements List |
| CMS | Code Management System |
| ConvFactors | Conversion_Factors (TLCSC) |
| CPDS | Computer Program Development Specification |
| CPPS | Computer Program Product Specification |
| CSC | Computer Software Component |
| CSCI | Computer Software Configuration Item |
| CVMA | Coordinate_Vector_Matrix_Algebra (TLCSC) |
| DACS | Defense Analysis Center for Software |
| DBMS | Data Base Management System |
| DCL | DIGITAL Command Language |
| DDD | Detailed Design Document |
| DEC | Digital Equipment Corporation |
| DMA | Direct Memory Access |
| DoD | Department of Defense |

| | |
|---|---|
| DoD-STD | Department of Defense Standard |
| DPSS | Digital Processinr, Subsystem |
| DSR | Digital Standard Runoff |
| DTM | DEC /Test Manager |
| FMS | Forms Management System |
| FORTRAN | FORmula TRANslation |
| GPMath | General_Purpose_Math (TLCSC) |
| HOL | Higher-Order Language |
| Hr | Hour |
| I/O | Input/Output |
| ISA | Inertial Sensor Assembly |
| JOVIAL | Jules Own Version of International Algebraic Language |
| LISP | List Processing (language) |
| LLCSC | Lower Level Computer Software Component |
| LOC | Lines of Code |
| MDAC | McDonnell Douglas Astronautics Company |
| MDAC-HB | McDonnell Douglas Astronautics Company - Huntington Beach |
| MDAC-STL | McDonnell Douglas Astronautics Company - St. Louis |
| MDC | McDonnell Douglas Corporation |
| MIL-STD | Military Standard |
| MRASM | Medium Range Air-to-Surface Missile |
| NM | Nautical Miles |
| NPNav | North_Pointing_Navigation_Parts (TLCSC) |
| OCU | Operator Control Unit |
| Opns | Operations |
| PC | Personal Computer |
| PCS | Parts Composition System |
| PDL | Program Design Language |
| R&D | Research and Development |
| RT | Remote Terminal |
| RTE | Real-Time Embedded |
| SDF | Software Development File |
| SDI | Strategic Defense Initiative |
| SDN | Software Development Notebook |
| SDR | Software Discrepancy Report |
| SEAFAC | System Engineering Avionics Facility |
| SEI | Software Engineering Institute |

| | |
|---|---|
| SEP/SCP | Software Enhancement Proposal/Software Change Proposal |
| SIGAda | Special Interest Group on Ada |
| SRS | Software Requirements Specification |
| STARS | Software Technology for Adaptable, Reliable Systems |
| stmt | statement |
| SURMOS | Start-Up Real-time Multi-tasking Operating System |
| TLCSC | Top-Level Computer Software Component |
| TLDD | Top-Level Design Document |
| UnivConst | Universal_Constants (TLCSC) |
| VAX | Virtual Address Extension |
| VMS | Virtual Memory System |
| WGS72 | World Geodetic System, 1972 |

# SECTION I

# INTRODUCTION

## 1. PURPOSE

This report contains a description of the work performed, the results achieved, and the lessons learned on the 11th Missile Application of the Common Ada Missile Packages Phase 2 (CAMP-2) project. CAMP-2 was a multi-year research effort in which the McDonnell Douglas Astronautics Company-St. Louis (MDAC-STL) demonstrated the feasibility and value of reusable Ada software parts in embedded, real-time, mission-critical, DoD applications. This was accomplished by (a) building a library of efficient and reusable Ada parts for missile flight applications, (b) building a prototype parts composition system (PCS), and (c) testing the parts and the PCS by using them on an actual missile application (the 11th Missile).

The CAMP project has been sponsored by the Air Force Armament Laboratory at Eglin Air Force Base, and partially funded by the Air Force Armament Division; the DoD Software Technology for Adaptable, Reliable Systems (STARS) Program Office; and the Air Force Electronic Systems Division. The Ada Joint Program Office (AJPO) sponsored the initial distribution of CAMP Ada parts to 120 Government agencies and contractors. This software is now available through the Air Force Defense Analysis Center for Software (DACS) at Griffiss Air Force Base, New York.

## 2. GOALS AND OBJECTIVES

The overall goal of CAMP-2 was to demonstrate the technical feasibility and value of reusable Ada missile parts and a PCS by building and using them on a realistic application. The 11th Missile Application involved the construction of an actual missile application using the Ada parts and the PCS, and testing of the developed system in a MIL-STD-1750A hardware-in-the-loop simulation. The initial goals of the application are enumerated below.

1. Construct a complete missile application using CAMP parts and the PCS, and test it in a 1750A hardware-in-the-loop simulation.

2. Evaluate the suitability of the CAMP parts and the PCS for real-time embedded missile applications.

3. Test the CAMP parts and the PCS, and recommend corrections and improvements.

4. Quantify the productivity improvement attributable to the use of CAMP parts and the PCS.

Although not explicitly stated as goals, the 11th Missile Application also served as the basis for (1) an evaluation of the suitability of Ada for real-time embedded missile applications, and (2) an evaluation of the suitability of an Ada/1750A compiler for real-time embedded applications.

## 3. DELIVERABLES

The deliverable products of the 11th Missile Application were:

1. Software Requirements Specification: The requirements of the missile application documented in accordance with DOD-STD-2167, AFATL-TR-88-24, Volume 1.

2. Top-Level Design Document: The architectural design for the 11th Missile system documented in accordance with DOD-STD-2167, AFATL-TR-88-24, Volume 2.

3. Test Plan: The plan by which the 11th Missile system was tested in accordance with DOD-STD-2167, AFATL-TR-88-22.

4. Test Report: The results of testing the application in accordance with DOD-STD-2167. This includes an evaluation of the 11th Missile development.

## 4. ORGANIZATION OF THE REPORT

Due to the large amount of data to be discussed in this report, it has been divided into three volumes. The remaining sections of Voiume II are organized as follows.

- Section II describes the development and testing of the 11th Missile Application.

- Section III evaluates the CAMP parts and their suitability for real-time embedded missile applications.

- Section IV evaluates the Kalman Filter Constructor, which is part of the CAMP PCS, and its suitability for real-time embedded missile applications.

- Section V evaluates the suitability and effectiveness of the Ada language for real-time embedded missile applications.

- Section VI evaluates the suitability of an Ada/1750A compiler for real-time embedded missile applications and shows how to "work around" the problems encountered.

- Section VII contains conclusions and recommendations.

Volume I describes the development and testing of the CAMP parts and the PCS. Volume III describes the development and testing of the Armonics Benchmarks.

# SECTION II

# DEVELOPMENT AND TESTING OF 11TH MISSILE APPLICATION

## 1. WHAT IS THE 11TH MISSILE?

The CAMP parts and parts composition system (PCS) were designed and implemented following a domain analysis of ten missiles (see Table 1). To test the parts and PCS in a realistic setting, an "11th Missile" was held in reserve and a portion of its software was implemented using them.

**TABLE 1. THE CAMP DOMAIN**

| | |
|---|---|
| 1. | Flight Software for the Medium Range Air-to-Surface Missile (AGM-109H) |
| 2. | Flight Software for the Medium Range Air-to-Surface Missile (AGM-109L) |
| 3. | Strapdown Inertial Navigation Program for the Unaided Tactical Guidance Project |
| 4. | Guidance and Navigation Program for the Midcourse Guidance Demonstration |
| 5. | Flight Software for the Tomahawk Land Attack Missile (BGM-109A) |
| 6. | Flight Software for the Tomahawk Anti-Ship Missile (BGM-109B) |
| 7. | Flight Software for the Tomahawk Land Attack Missile (BGM-109C) |
| 8. | Flight Software for the Tomahawk Land Attack Missile (BGM-109G) |
| 9. | Flight Software for the Harpoon Missile (Block 1C) |
| 10. | Safeguard Spartan Missile |

The 11th Missile Application was based on a cruise missile application that was originally implemented in JOVIAL J73. The original application had five MIL-STD-1750A processors and an Inertial Sensor Assembly (ISA), which communicate by means of a MIL-STD-1553B data bus (see Figure 1). The shared memory contained terrain altitude data. The processors and their primary functions are shown in Table 2. In addition, all processors were programmed to perform the following support functions:

- Restart the application program
- Return control to start-up ROM
- Communicate via the 1553B bus
- Issue periodic status messages

The 11th Missile Application is a re-implementation (starting with a new requirements specification) of the navigation, ground alignment, Kalman filtering, ISA interface, lateral guidance, lateral-directional autopilot, and support functions of the original software. The navigation, Kalman filtering, lateral guidance, and lateral-directional autopilot were chosen because there were CAMP parts to support those functions. The other functions were required to complete a functioning computer program.

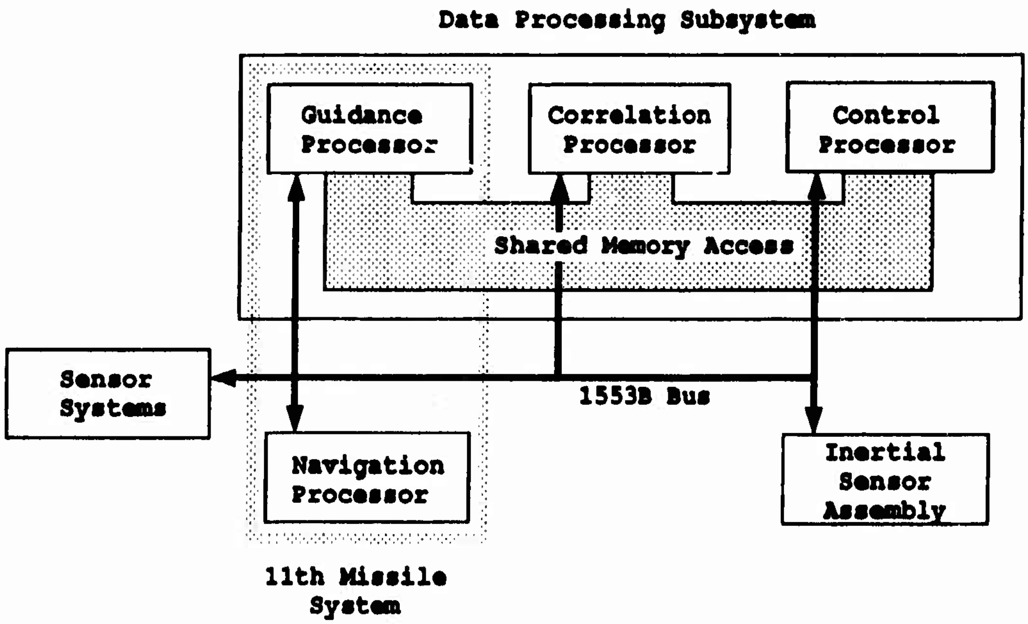


**Figure 1.** 11th Missile Hardware Design

## TABLE 2. PROCESSORS AND THEIR FUNCTIONS

| Processor | Functions |
|---|---|
| Control | Communicates with operator console<br>Downloads, starts, and stops software in all other machines<br>Mode logic |
| Navigation | Wander-azimuth navigation<br>Transfer alignment<br>Ground alignment<br>21-state Kalman filter<br>Start up, test, and communicate with ISA |
| Guidance | Waypoint-steering lateral guidance<br>Vertical guidance<br>Lateral-directional autopilot |
| Correlation | Dedicated to Terrain Profile Matching |
| Sensor | Controls sensor system hardware |

Two versions of the 11th Missile Application were written. The first version ("Parts Method") was written using the CAMP parts, but not the CAMP PCS. The second version ("PCS Method") used the PCS to generate Kalman filter code. The PCS Method implementation was not a complete rewrite of the code; rather, the PCS-generated Kalman filter code was integrated with the rest of the Parts Method code and unit tested.

## 2. ELEVENTH MISSILE DEVELOPMENT

### a. Requirements Development

The 11th Missile requirements were developed in accordance with DOD-STD-2167, and documented in a Software Requirements Specification (SRS). This specification combined elements of the existing application's navigation, guidance, and interface requirements (see Figure 2).

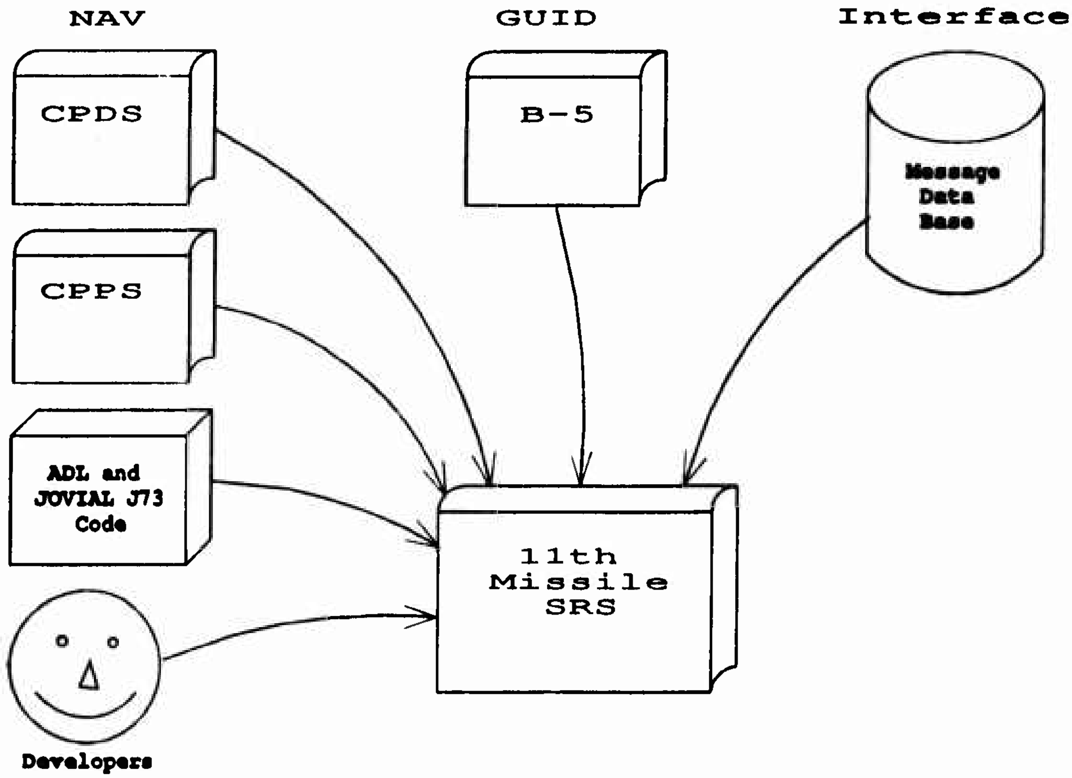


Figure 2. 11th Missile Requirements Came From Several Sources

### (1) Navigation Requirements

The 11th Missile navigation requirements came from several sources. The existing application's navigation requirements were documented in a MIL-STD-1679 Computer Program Development Specification (CPDS), but it was badly out-of-date; it served primarily as an outline of the high-level requirements. There was also a MIL-STD-1679 Computer Program Product Specification (CPPS), which included Ada Design Language (ADL) and which was more nearly up-to-date. Interviews with the original software developers were necessary to determine which requirements had changed since the CPPS; in these cases, the requirements were abstracted from the updated ADL or, occasionally, from the JOVIAL code.

### (2) Guidance Requirements

The guidance requirements for the existing application were specified in a MIL-STD-483 B-5 development specification. The lateral guidance and lateral-directional autopilot algorithms were reused from the Medium Range Air-to-Surface Missile (MRASM) program, therefore, these requirements were stable and the specification was up-to-date and complete.

The 11th Missile team discovered an error in the autopilot requirements while developing the open-loop integration test for the Guid Computer. The problem was referred to the requirements group for the existing application; they verified the error and corrected their requirements specification. The CAMP project corrected the 11th Missile SRS to conform to the revised requirements.

### (3) Interface Requirements

The 1553B bus protocols and the formats of all bus messages were specified in a database maintained by the original application. That project used the database to automatically generate the JOVIAL code that specified the message formats for the application programs, the FORTRAN code for the real-time-simulation software, and the interface requirements specification. The 11th Missile team used the database to get up-to-date message specifications and change notices.

## b. Top-Level Design

### (1) Object-Oriented Design

In an object-oriented design, the requirements are functionally decomposed and assigned to Ada packages. There may be several levels of decomposition, which generally correspond to nested Ada packages.

In the 11th Missile implementation of this method, tasks are subsidiary to packages. Tasks were generally not defined in a high-level package specification. If a task had to be invoked from outside the high-level package, an interface procedure to call the task was provided. Tasks were used primarily for control; they called on packages to execute the controlled functions. A good example of this type of design is the Kalman_Filter package (see Figures 3 and 4).

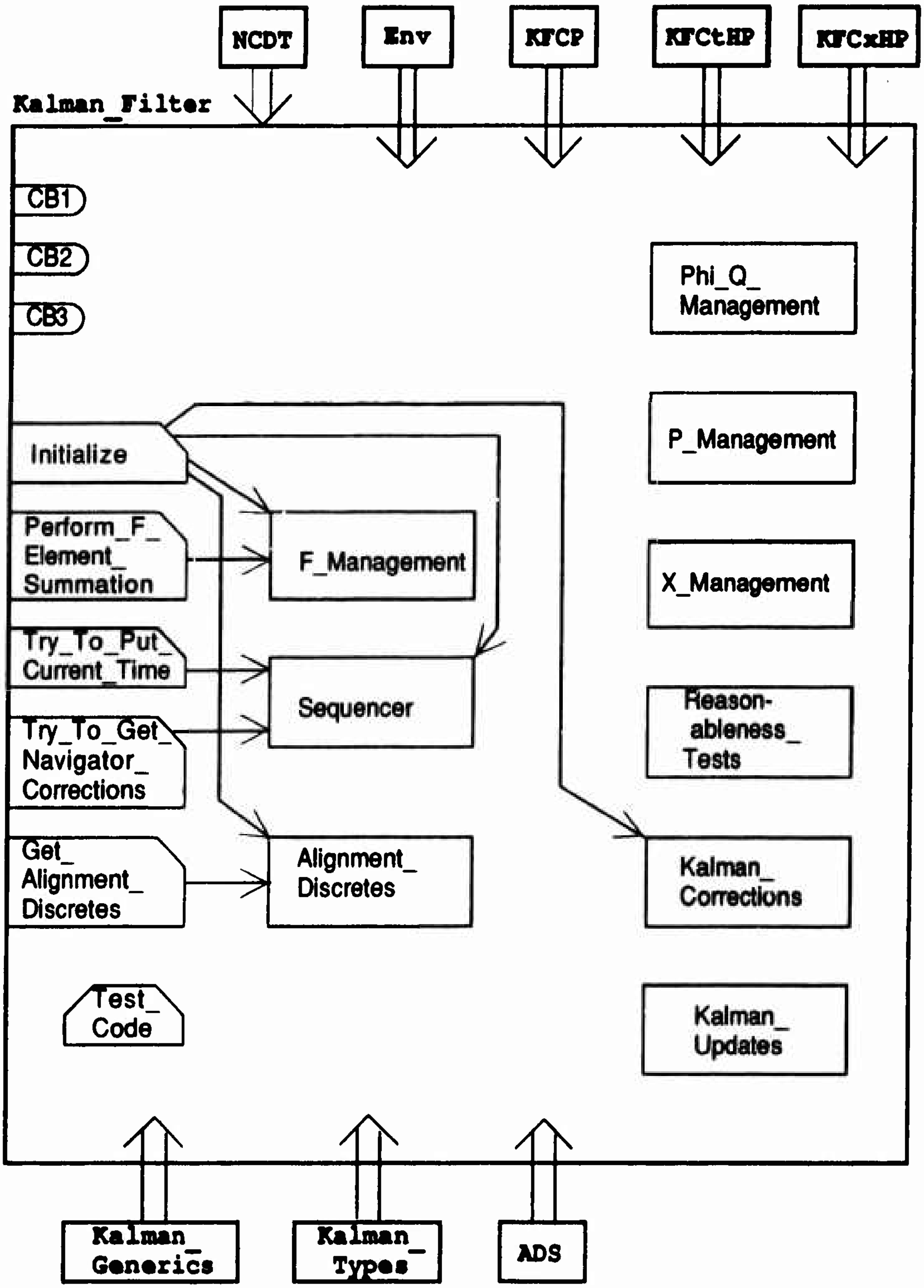


**Figure 3.** Package Kalman_Filter Structure Chart

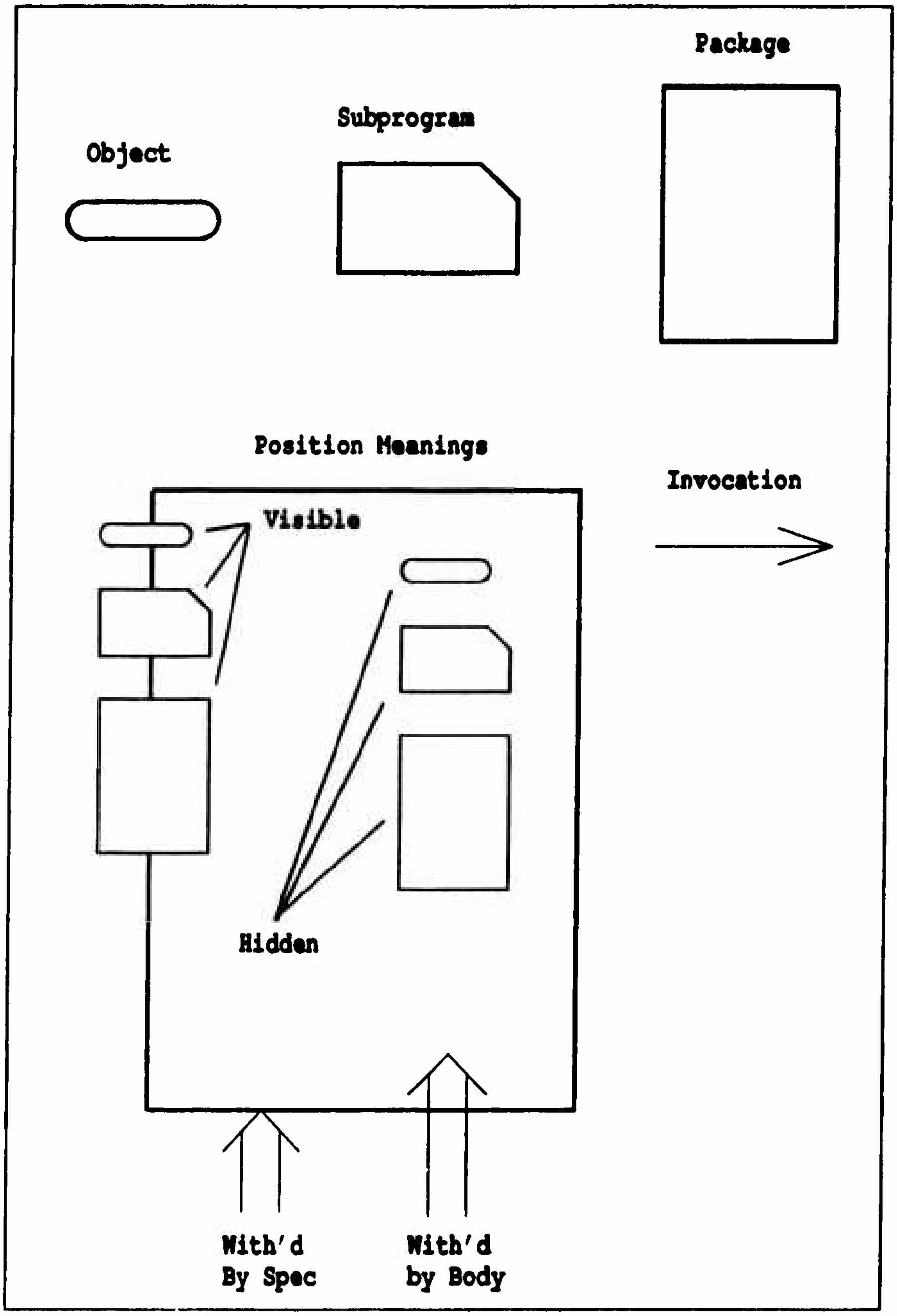


Figure 4. Legend for Kalman_Filter Structure Chart

This philosophy was not always followed. For example, the BIM_Interface package was written before these design decisions were made, and contains a large task which is directly visible and which does its own processing.

Tasks cannot be completely ignored at the system level, however. The architectural design had to specify the processing priority of each task.

### (2) The Architectural Design Process

In this section, the Navigation Processor design serves as an example of the architectural design process. The two major LLCSCs of the Navigation TLCSC are Environment and Nav_Software (see Figures 5 and 6). The Environment LLCSC provides the interface to all external devices, while Nav_Software performs the navigation, alignment, quaternions, and Kalman filtering functions.

The first step in the architectural design was to go through the SRS and determine all the places where CAMP parts could be used. An annotated copy of the SRS showed where each part applied. The resulting parts list was used to drive the architectural and detailed designs, and maximize reuse of existing parts.

The next step was to "rough out" the decomposition diagram (see Figures 5 and 6), structure charts, and the Ada package specifications. See Figure 3 for a sample structure chart. Different approaches were used for the two major LLCSCs. The Environment LLCSC was implemented as a single package; the Nav_Software LLCSC was implemented as five packages, since a single package would have been too large.

The top-level design went through several iterations. In general, each iteration hid more data, i.e. material moved from the package specification to the body. Also, as the design developed, lower-level packages were created and the required functions were mapped to them. The end result of this process was Ada code for the package and task specifications, skeleton Ada code for the task bodies, the top-level decomposition diagram (see Figures 5 and 6), and a series of structure charts.

The architecture was informally reviewed by the design group as it developed. The CAMP Program Manager reviewed the architecture twice; these reviews concentrated on the decomposition diagram and the structure charts. Formal walkthroughs of the high-level Ada code followed the final management review.

## c. Detailed Design and Code

Detailed design and coding were combined into one phase. In some respects this phase was a continuation of the top-level design process.

Each high-level package was assigned to a single designer, who was responsible for the detailed design, code, and headers. All design/code was walked through by the entire 11th Missile team. There was at least one walkthrough for each package; the larger packages (e.g., Environment, Kalman_Filter) were broken down and underwent several walkthroughs. The walkthroughs sought to ensure that the code met the requirements, interfaced properly with other code, conformed to project standards, and had complete headers.

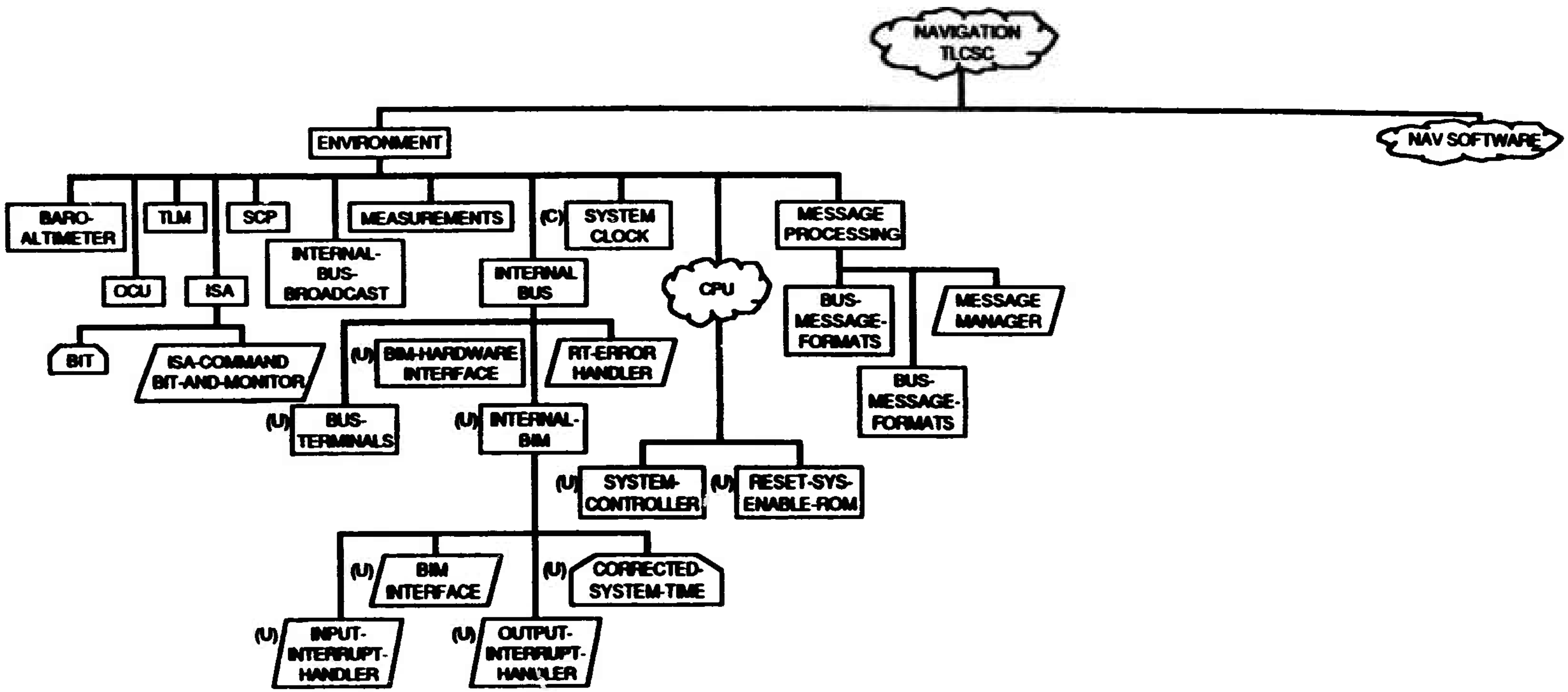


**Figure 5. NAV Computer Top-Level Decomposition: Environment**

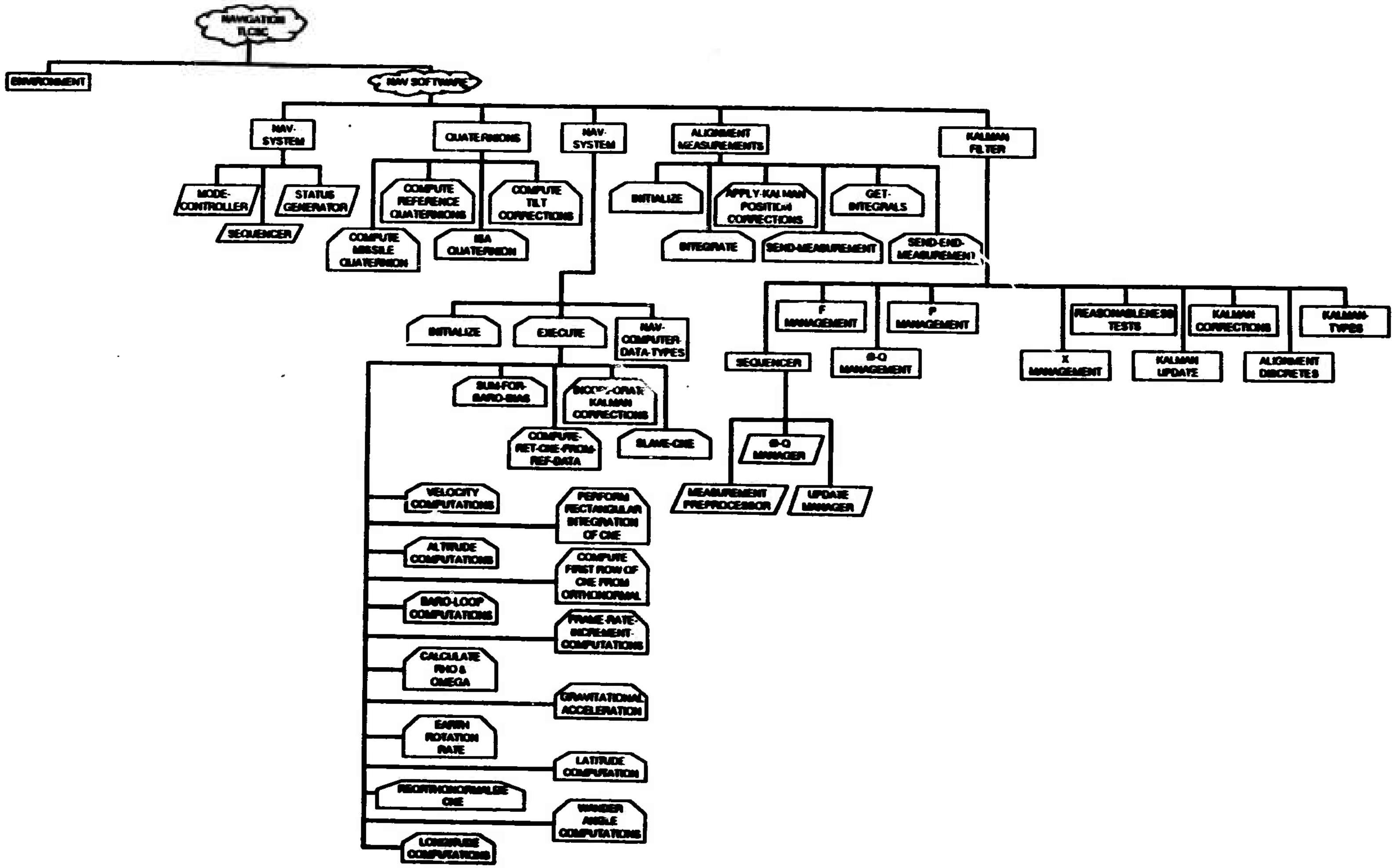


**Figure 6.** NAV Computer Top-Level Decomposition: Nav_Software

### d. Unit and Component Level Test Methods

In general, the 11th Missile team tested both units and packages (components). Usually unit- and package-level tests were separate, but occasionally they were combined or the package-level test was skipped. Table 3 presents the decision matrix that was used to determine the level of testing that was required.

**TABLE 3. UNIT- AND PACKAGE-LEVEL TEST DECISION MATRIX**

| | Case | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| Condition | | | | | |
| Unit | - | Simple | Simple | Complex | Complex |
| Interaction | None | Simple | Complex | Simple | Complex |
| Tests Required | | | | | |
| Unit | Y | Combine | Part of package test | Y | Y |
| Package | N | | Y | Part of unit tests | Y |

Package-level testing was not required if there was no interaction between the units in the package. For example, Kalman_Types is a collection of data type definitions and operators; no data is stored in the package body and the units do not invoke each other.

If the units and the interactions between them were both simple, the unit tests were folded into the package-level test (e.g., Alignment_Measurements) or the unit tests covered the package-level test requirements (e.g., Quaternions).

If the units were complex, then unit-level tests were always required. If the unit interactions were simple, the package-level test requirements could be covered by the unit tests (e.g., BIM_Interface); if complex, a separate package-level test was required (e.g., Kalman_Filter).

The CAMP parts were assumed to be correct and were not tested separately. They were tested indirectly as part of the units that invoked them.

#### (1) Unit Test Approach

The 11th Missile team designed unit tests to cover both *white box* and *black box* viewpoints. A *white box* test is designed with knowledge of the unit's structure. The test cases are set up to exercise all paths through the unit and to invoke all branch conditions. A *black box* test is a functional test that assumes nothing about the unit's internal structure. It passes in a representative sample of input data and checks to see if the output is as expected. The Alignment_Measurements package will be used to illustrate these two approaches to unit testing.

The Alignment_Measurements package (see Figure 7) takes a sequence of integrated velocities from the navigator, keeps and corrects running sums of them, and periodically formats the sums

into measurements and sends them to package Kalman_Filter. Calls to control procedures (Initialize, Set_Measurement_Time, and Cancel_Measurement) initialize the package, specify when a measurement is to be sent to Kalman_Filter, and occasionally cancel a measurement. Integrate, Put_Reference_Velocity_Integrals, and Apply_Kalman_Position_Corrections receive data needed to compute or correct the integrated-velocity sums. Get_Integrals returns the current integrated-velocity sums.

```
with Nav_Computer_Data_Types;
package Alignment_Measurements is

   package NCDT renames Nav_Computer_Data_Types;

   procedure Initialize (Initial_Time       : in NCDT.Seconds;
                         Reference_Altitude : in NCDT.Feet_FP;
                         Velocity           : in NCDT.Velocity_Vectors);

   procedure Integrate (Eff_Time_Of_Incr_Data : in NCDT.Seconds;
                        Velocity              : in NCDT.Velocity_Vectors;
                        Altitude              : in NCDT.Feet_EFP);

   procedure Put_Reference_Velocity_Integrals
               (X_Velocity_Integral : in NCDT.Feet_FP;
                Y_Velocity_Integral : in NCDT.Feet_FP);

   procedure Apply_Kalman_Position_Correction
               (Position_Error_X : in NCDT.Earth_Position_Radians;
                Position_Error_Y : in NCDT.Earth_Position_Radians);

   procedure Get_Integrals (Integrated_Vel_X : out NCDT.Feet_FP;
                            Integrated_Vel_Y : out NCDT.Feet_FP);

   procedure Set_Measurement_Time (Time : in NCDT.Seconds);

   procedure Cancel_Measurement;

end Alignment_Measurements;
```

**Figure 7.** Package Alignment_Measurements

A black box test would invoke the package with a sequence of control and data calls, verify that the current sums returned by Get_Integrals are correct, verify that the package sends correct measurements to Kalman_Filter at the correct times, and verify that a measurement is not sent if it has been cancelled. The test designer would use the requirements specification and the Ada package specification to develop the tests.

The white box test designer would also use the package and procedure body listings to generate test cases that cover all the paths and exercise all the branch conditions. For example, the white box test of procedure Cancel_Measurement (see Figure 8) would call this procedure twice, once with measurement_pending true and once with it false. (Measurement_Pending is stored in the package body.)

```
with Environment;
separate (Alignment_Measurements)
procedure Cancel_Measurement is
begin

   if Measurement_Pending then

--     --send invalid measurement to the Kalman
       Environment.Measurements.Here_is_Alignment_Measurement
          (Time_of_Alignm_Data => Measurement_Time,
           Alignm_Valid        => FALSE,
           MeasX               => 0.0,
           MeasY               => 0.0,
           MeasZ               => 0.0,
           VarX                => 0.0,
           VarY                => 0.0,
           VarZ                => 0.0);

--     --cancel measurement pending flag
       Measurement_Pending := FALSE;

   end if;

end Cancel_Measurement;
```

**Figure 8.** Procedure Cancel_Measurement

The Alignment_Measurements test was primarily a black box (i.e., functional) test, with some additional test cases to cover the white box criteria.

### (2) Process

After a unit had been walked through and approved, the unit's designer wrote the test procedure. Procedures were written in DoD-STD-2167 format, and each one was reviewed by the entire 11th Missile team. An engineer other than the unit's designer wrote the test driver code and executed the test.

Most unit and package tests were first executed on the VAX using the DEC Ada compiler (see Figure 9). This approach was originally adopted because of the problems encountered with the early Ada/1750A cross-compilers. There was an unexpected productivity benefit, however, because DEC's program debugging and library management tools were much better than those provided by the 1750A cross-compiler. Errors could be found and corrected much faster with DEC's tools than with the cross-compiler's tools. These units were then tested on a 1750A simulator or 1750A hardware. Since the code and the test drivers had been checked out, these tests were primarily to debug the 1750A cross-compiler, time the software, and check numerical accuracy.

Some units (e.g., the 1553B bus interface) could not be meaningfully tested unless they were compiled by a 1750A-targeted compiler, and so were first tested on the 1750A Simulator.

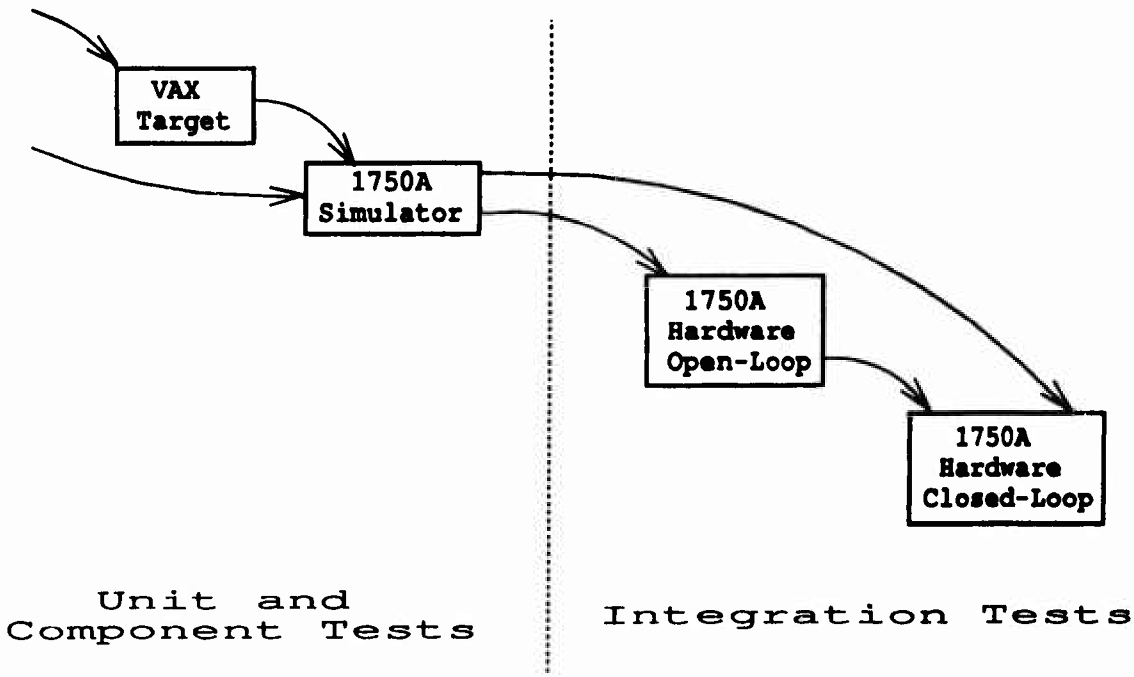


**Figure 9.** Test Approach

## e. Integration and Hardware-in-the-Loop Tests

The Laser Guidance group of McDonnell Douglas Astronautics Company developed the simulation facility used by the 11th Missile Application; the 11th Missile Application software was developed to be consistent with the requirements of this facility. The hardware configurations used in integration and hardware-in-the-loop tests are shown in Figure 10. The baseline simulation setup, Operator Control Unit (OCU) and flight Digital Processing Subsystem (DPSS), utilized two 1750A processors. Because only one 1750A processor was available, the alternate breadboard/monitor configuration was used. With this alternate configuration, the 11th Missile Application software required minor modifications. The Guidance computer was converted from a remote terminal (RT) to a bus controller (BC).

The hardware-in-the-loop tests for the Guidance CSC were successfully completed, but the Navigation CSC hardware-in-the-loop testing could not be performed given the inability of the Ada/1750A cross-compiler to generate correct Ada code for this portion of the application and the unavailability of work-arounds. The tests uncovered six errors in the 11th Missile Application software and three errors in the Ada/1750A compiler code. The Guidance CSC required 87.2% of the throughput. A more detailed description of the results may be found in the Software Test Report (Reference 1).

The 11th Missile Application testing demonstrated that given efficient and effective Ada com-

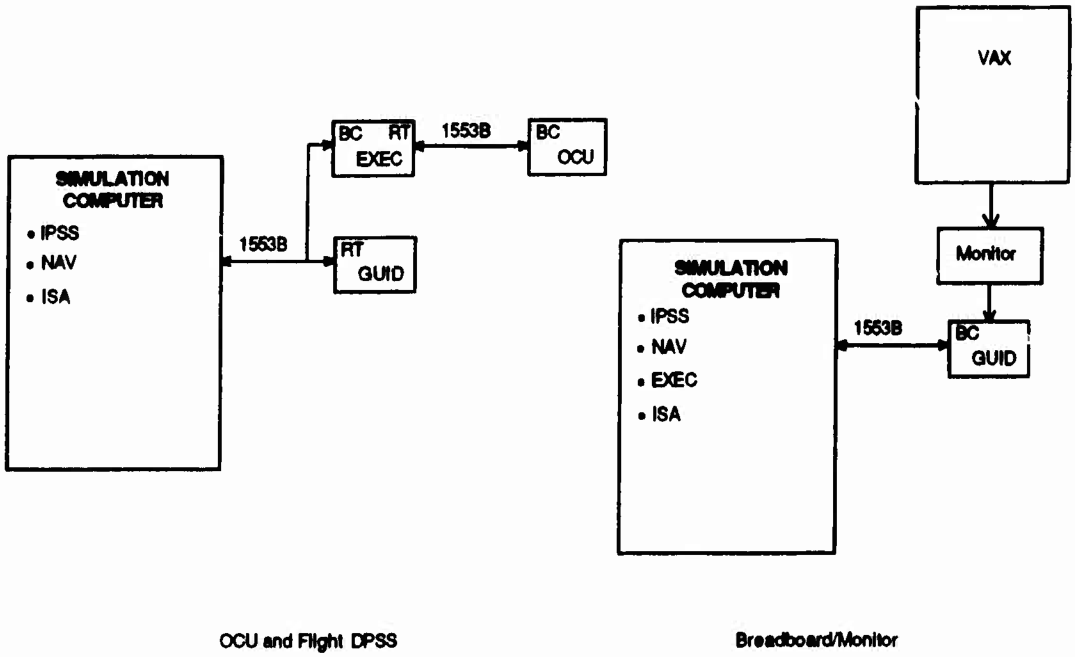


**Figure 10. Hardware Configurations**

pilers, the CAMP parts may be used in embedded real-time software. The CAMP parts were functionally correct, but, with the current Ada/1750A compilers, most generics had to be manually instantiated (see Section VI).

The 11th Missile Application CSCI is not effective flight software due to long execution time. The Guidance CSC runs three times slower than a nearly equivalent JOVIAL version of the same application. The Navigation CSC would not have been able to run in real time. The main sources of inefficiencies were Ada task rendezvous and the compiler's implementation of generics; Section VI discusses the Ada issues in more detail.

### f. Tools

Table 4 lists the tools used by the 11th Missile team and the software development phases in which they were used.

**TABLE 4. TOOLS USED BY SOFTWARE DEVELOPMENT PHASE**

| Tool | Top Level Design | Detailed Design | Unit Test | Integration Test |
|---|---|---|---|---|
| Requirements Maps | X | | | |
| Message Maps | X | X | | |
| Decomposition Diagrams | X | | | |
| Structure Charts | X | | | |
| Digital Standard Runoff | X | | X | |
| Comment Extractor | X | | | |
| DEC Ada Compilation System | X | X | X | |
| TLD VAX/1750A Ada Compiler System | | | X | |
| Compiler Conversion Utilities | | X | X | |
| MDAC-HB 1750A Simulator | | | X | |
| MIKROS 1750A Emulator | | | X | |
| Hardware-In-The-Loop Simulation | | | | X |
| DEC Code Management System | | X | X | |
| Software Development Files | | X | X | |
| Development Status Database | | X | X | |
| Smart Code Counter | | X | X | |

### (1) Requirements Mapping Tools

Requirements maps were used to verify the completeness of the design. These maps (see Table 5 for an example) correlated the software implementation and the requirements; they were used to verify that all requirements were implemented. The maps also appeared in the TLDD. The message maps (see Table 6 for an example) showed which element (subprogram or task) processed each 1553B bus message, and served to verify that the software processed every bus message.

### (2) Design Visualization Tools

Decomposition diagrams and structure charts were used to provide two different views of the software design. The decomposition diagram clearly showed the functional partitions of the software; the structure charts showed the packaging structure of the Ada code.

The two visual aids are consistent, except for the data type definition packages. For example, package Kalman_Types is logically a part of the Kalman Filter function, and is therefore shown below package Kalman_Filter in the decomposition diagram (see Figure 6, at the far right edge). However, this would have forced the Kalman_Filter and Environment packages to "with" each other, which would violate an Ada language rule. Therefore, package Kalman_Types is separate from Kalman_Filter and appears as such in its structure chart (see Figure 3).

## TABLE 5. NAVIGATION TLCSC FUNCTIONAL ALLOCATION TO LOWER LEVEL COMPUTER SOFTWARE COMPONENTS

(Part 1 of 2)

| STLDD ALLOCATION | | | CSCI REQUIREMENT | |
|---|---|---|---|---|
| LLCSC NUMBER | LLCSC NAME | PARAGRAPH NUMBER | REQUIREMENT NAME | PARAGRAPH NUMBER |
| | Environment | | | |
| | Barometric_Altimeter | | Barometric Altimeter Programmed Input Interface | 3.3.3.2.3 *2 |
| | | | Altitude Retrieval Subfunction | 3.4.2.4 *2 |
| | | | Altimeter Subfunction - Scaling | 3.4.5.1.1 *2 |
| | OCU | | 1553B Bus Messages Subfunction (Input) | 3.4.5.1.3 *1*3 |
| | | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | TLM | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | ISA | | 1553B Bus Messages Subfunction (Input) | 3.4.5.1.3 *1*3 |
| | | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | | | Monitor Subfunction | 3.4.7.5 *2 |
| | SCP | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | Internal_Bus_Broadcast | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | Measurements | | 1553B Bus Messages Subfunction (Input) | 3.4.5.1.3 *1*3 |
| | | | 1553B Bus Message Tables | App. 1 *1 *3 |
| | Internal_Bus | | 1553B Bus Interface Module (BIM) Interface | 3.3.3.2.1 *1*2 |
| | | | Interrupt Management Function | 3.4.1 *1 *2 |
| | | | 1553B Input Message Handler Subfunction | 3.4.2.1 *1 *2 |
| | | | 1553B Output Message Handler Subfunction | 3.4.2.2 *1 *2 |
| | | | 1553B Management - Reinitialization | 3.4.2.3 *1 *2 |
| | | | 1553B Management - Error Handling | 3.4.2.3 *1 *2 |

*1 Satisfies those parts of the paragraph that apply to the Navigation Computer
*2 This is the primary CSC to meet the requirements of this section.
*3 This is a secondary CSC; it supports the primary CSC in meeting the requirements of this section.

**TABLE 5. NAVIGATION TLCSC FUNCTIONAL ALLOCATION TO LOWER LEVEL COMPUTER SOFTWARE COMPONENTS**

**(Concluded)**

| STLDD ALLOCATION | | | CSCI REQUIREMENT | |
|---|---|---|---|---|
| LLCSC NUMBER | LLCSC NAME | PARAGRAPH NUMBER | REQUIREMENT NAME | PARAGRAPH NUMBER |
| | System_Clock | | Master Time Subfunction | 3.4.3.2 *1 *2 |
| | | | System Data Subfunction - Master Time | 3.4.5.1.2 *1*2 |
| | CPU | | Start-Up Real-Time Multi-Tasking Operating System (SURMOS) | 3.3.3.2.2 *1*2 |
| | | | System Control Function | 3.4.4 *1 *2 |
| | Message Processing | | Local Time Subfunction | 3.4.3.1 *1 *2 |
| | | | 1553B Bus Messages Subfunction (Input) | 3.4.5.1.3 *1*2 |
| | | | 1553B Bus Messages Subfunction (Output) | 3.4.5.2.1 *1*2 |
| | | | 1553B Bus Message Tables | App. 1 *1 *2 |
| | Nav Software | | | |
| | Nav_System | | System Data Subfunction - System Mode | 3.4.5.1.2 *2 |
| | | | Status Message | 3.4.5.2.2 *1*2 |
| | | | Navigation Function - Mode Control | 3.4.7.0 *2 |
| | Navigation_Operations | | Altimeter Subfunction - Bias Computation | 3.4.5.1.1 *2 |
| | | | (Navigator Subfunction) Process Control | 3.4.7.1.2 *2 |
| | | | (Navigator Subfunction) Processing | 3.4.7.1.3 *2 |
| | Alignment_Measurements | | Alignment Subfunction | 3.4.7.3 *2 |
| | Kalman_Filter | | Telemetry | 3.4.5.2.3 *1*2 |
| | | | Kalman Filter Subfunction | 3.4.7.2 *2 |
| | Quaternions | | ISA Initialization Subfunction | 3.4.7.4 *2 |

*1 Satisfies those parts of the paragraph that apply to the Navigation Computer
*2 This is the primary CSC to meet the requirements of this section.
*3 This is a secondary CSC; it supports the primary CSC in meeting the requirements of this section.

# TABLE 6. NAV COMPUTER MESSAGE MAP

(Part 1 of 2)

| Message ID | Msg Symbol | Handled By |
|---|---|---|
| IPSS-NAV - 40 | KULFU | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 41 | KUNTPF2D | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 42 | KUNTPF3D | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 43 | KUPTH | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 45 | TPMUPD | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 50 | LFUTIME | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 51 | NTPF2DTIME | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 52 | NTPF3DTIME | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 53 | PTHTIME | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| IPSS-NAV - 54 | TPMTIME | Kalman_Filter.Sequencer.Measurement_Preprocessing |
| ISA -BRD -132 | APDATA | Nav_System.Sequencer |
| ISA -BRD -133 | CLOCK | Environment.Message_Management.Message_Manager |
| ISA -NAV - 8 | INCNAV | Nav_System.Sequencer |
| ISA -NAV - 9 | DYNAM | Nav_System.Sequencer |
| ISA -NAV - 10 | ISSTAT | Environment.ISA.ISA_Comm_BIT_And_Monitor |
| NAV -BRD -130 | NAVDAT | Nav_System.Sequencer (via Navigation_Operations.-<br>Execute_Navigator) |
| NAV -BRD -131 | NAVPOSCOV | Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages |
| NAV -EXEC- 1 | ATTITEXEC | Nav_System.Sequencer |
| NAV -GUID- 1 | ATTITGUID | Navigation.Sequencer |
| NAV -GUID- 2 | ATTITGUID2 | Navigation.Sequencer |
| NAV -ISA - 3 | INITAT | Nav_System.Mode_Controller |
| NAV -ISA - 4 | FRINC | Nav_System.Sequencer (via Navigation_Operations.-<br>Execute_Navigator) |
| NAV -ISA - 5 | KALCOR | Kalman_Filter.Sequencer.Update_Manager.Reinitialize<br>(initial tilt correction message)<br>Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages (all others) |
| NAV -ISA - 6 | ISACMD | Environment.ISA.ISA_Comm_BIT_And_Monitor |
| NAV -OCU - 4 | NAVSTAT | Nav_System.Status_Generator |
| NAV -OCU - 6 | RTBIMERR | Environment.RT_BIM_Error_Handler |
| NAV -SCP - 2 | ALPHA | Nav_System.Sequencer |
| NAV -SCP - 3 | CORRECT | Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages |
| NAV -TLM - 7 | KALRLT2 | Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages |
| NAV -TLM - 13 | KF_DIAG | Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages (P and X terms)<br>Kalman_Filter.Sequencer.Phi_Q_Manager (phi & Q terms) |
| NAV -TLM - 15 | KALGAN | Kalman_Filter.Sequencer.Update_Manager.-<br>Update_And_Send_Messages |
| NAV -TLM - 16 | VAR | Kalman_Filter.Sequencer.Update_Manager.-<br>Retrieve_And_Propagate (P props)<br>Kalman_Filter.Sequencer.Update_Manager.-<br>Update_And_Send_Messages (P updates) |
| NAV -TLM - 17 | PHIDIAGNOS | Kalman_Filter.Sequencer.Phi_Q_Manager (phi props)<br>Kalman_Filter.Sequencer.Update_Manager.-<br>Retrieve_And_Propagate (P & X props) |
| NAV -TLM - 18 | KALRLT1 | Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages |
| NAV -TLM - 19 | KALSTER | Kalman_Filter.Sequencer.Update_Manager.-<br>Retrieve_And_Propagate (X props)<br>Kalman_Filter.Sequencer.Update_Manager.-<br>Update_And_Send_Messages (X updates)<br>Kalman_Filter.Sequencer.Update_Manager.-<br>Send_Correction_Messages (cum errors) |

TABLE 6. NAV COMPUTER MESSAGE MAP

(Concluded)

```
Message ID       Msg Symbol   Handled By
----------       ----------   ----------

NAV -TLM - 21    QDIAGNOST    Kalman_Filter.Sequencer.Phi_Q_Manager (Q props)
                              Kalman_Filter.Sequencer.Update_Manager.-
                               Retrieve_And_Propagate (P props)
NAV -TLM - 23    FDTDIAGNOS   Kalman_Filter.Sequencer.Phi_Q_Manager
NAV -TLM - 26    KFMEAS       Kalman_Filter.Sequencer.Update_Manager.-
                               Update_And_Send_Messages
NAV -TLM - 28    FSUMDIAG     Kalman_Filter.Sequencer.Phi_Q_Manager (phi & Q props)
NAV -TLM - 33    KALPER       Kalman_Filter.Sequencer.Update_Manager.-
                               Update_And_Send_Messages via
                                Kalman_Filter.Alignment_Discretes.Put_P
OCU -NAV -  2    NAVCMD       Nav_System.Mode_Controller
OCU -NAV - 82    PREP_DWNLD   Environment.Message_Manager via System_Controller
OCU -NAV - 84    START_APPL   Environment.Message_Manager via System_Controller
OCU -NAV - 86    INTGNDALN    Nav_System.Mode_Controller
OCU -NAV - 87    INDGNDALN    Nav_System.Mode_Controller
OCU -NAV - 92    NEWLEVARM    Nav_System.Mode_Controller
SCP -NAV - 39    DOP_TIME     Kalman_Filter.Sequencer.Measurement_Preprocessing
SCP -NAV - 40    DOPUPD       Kalman_Filter.Sequencer.Measurement_Preprocessing
TPM -NAV - 45    TPMUPD       Kalman_Filter.Sequencer.Measurement_Preprocessing
```

### (3) Documentation Tools

The 11th Missile team used Digital Standard Runoff (DSR) to format the top-level design document (TLDD) and the test procedures document. The bulk of the TLDD was extracted from the code headers and formatted for DSR by the Top-Level Design Comment Extractor (see Volume I, section II.2.e(4)), a tool developed by MDAC-STL.

### (4) Ada Compilers

The 11th Missile team had access to two different VAX/1750A cross-compilers; these will be referred to as "Compiler A" and "Compiler B". This is partly to protect the compiler vendors' proprietary information, and partly because the information presented here will (hopefully) soon be out-of-date as improvements are made to the compilers.

Neither compiler handled generics well enough to use the CAMP parts without modification (see Volume I, section VI). For this reason, and because DEC's Ada Compilation System (ACS) provides far better library management and code debugging tools, the 11th Missile team used the DEC Ada compiler extensively. The DEC compiler was the only one used during the design phases, and was used with a 1750A cross-compiler during unit testing (see Section II.2.d). In the top-level design phase, it was particularly useful when designing the task bodies, since the compiler automatically checked task *accept* statements and subprogram calls for consistency with the task *entry* and subprogram definitions, respectively. In both design phases, code had to compile before it was walked through. "Compiler B" was used for the 1750A version of the unit tests and for the integration tests.

Using three Ada compilers meant developing three versions of some of the code. The compiler-specific code consisted primarily of pragmas and representation specifications. The 11th Missile team did this by writing all three versions in one file and commenting out the lines that did not apply to the DEC compiler. Utility programs converted the files for use by another compiler by commenting out the DEC-specific statements and activating the statements for the specified compiler (see Figure 11).

### (5) Test Tools

Most unit tests were executed on the MDAC-HB 1750A Simulator, a FORTRAN program that simulates the operation of a 1750A chip. Using this program, the 11th Missile team was able to run those unit tests that did not use an external interface (i.e., the 1553B bus or the barometric altimeter port).

The simulator is quite slow, so some computation-intensive unit tests were executed on the MIKROS 1750A Emulator or on a 1750A breadboard from the original 11th Missile project. The MIKROS is a SEAFAC-certified 1750A implementation that is designed to be a co-processor with an IBM PC/AT. Programs were downloaded from the IBM PC to the 1750A target and executed on the 1750A hardware. The breadboard is built around an MDC281 implementation of the 1750A architecture and contains 128K of memory. For unit tests, software was downloaded and controlled from a VAX 11/780 via a 1750A Monitor.

The integration tests were executed on the hardware-in-the-loop simulation (see Section II.2.e).

### (6) Other Software Development Tools

Two other tools used during software development were Software Development Notebooks (SDNs) and the DEC Code Management System (CMS).

Generally, there was an SDN for each high-level package, but if it was large (e.g., Environment), there could be separate SDNs for its sub-packages. There were approximately forty 11th Missile SDN's. A module's Software Development Notebook (SDN) contained the following:

- The SRS pages that applied to the module
- The Ada code
- The test plan pages that applied to the module
- The test driver's code
- The test results

CMS is a configuration management tool, which served as the central source code library for the 11th Missile and kept track of code modifications. Only one person could reserve a file from CMS at a time, thus ensuring that one person's revisions could not be lost due to a file being overwritten by another person. CMS is integrated with the Ada compilation system (ACS) to the extent there is an ACS command to recompile all "out of date" modules (object code older than the corresponding source code in

**** DEC Version ****

```
        for initat_messages use
           record
              word_count     at  0*RP.storage_units_per_word range  0..15;
              source         at  1*RP.storage_units_per_word range  0.. 3;
              destination    at  1*RP.storage_units_per_word range  4.. 7;
              message_number at  1*RP.storage_units_per_word range  8..15;
              effective_time at  2*RP.storage_units_per_word range  0..31;
--"B"         init_quat_0    at  4*RP.storage_units_per_word range  0..15;
--"A"         init_quat_0    at  4*RP.storage_units_per_word range  0..15;
--"A"         init_quat_1    at  5*RP.storage_units_per_word range  0..15;
--"B"         init_quat_1    at  5*RP.storage_units_per_word range  0..15;
          end record;

        for initat_messages'size use BI.message_size; --VAX
--"B"   for initat_messages'size use BI.message_size+32;
--"A"   for initat_messages'size use BI.message_size;
```

**** "B" Version ****

```
        for initat_messages use
           record
              word_count     at  0*RP.storage_units_per_word range  0..15;
              source         at  1*RP.storage_units_per_word range  0.. 3;
              destination    at  1*RP.storage_units_per_word range  4.. 7;
              message_number at  1*RP.storage_units_per_word range  8..15;
              effective_time at  2*RP.storage_units_per_word range  0..31;
              init_quat_0    at  4*RP.storage_units_per_word range  0..15; --"B"
--"A"         init_quat_0    at  4*RP.storage_units_per_word range  0..15;
--"A"         init_quat_1    at  5*RP.storage_units_per_word range  0..15;
              init_quat_1    at  5*RP.storage_units_per_word range  0..15; --"B"
          end record;

--VAX   for initat_messages'size use BI.message_size;
        for initat_messages'size use BI.message_size+32; --"B"
--"A"   for initat_messages'size use BI.message_size;
```

**** "A" Version ****

```
        for initat_messages use
           record
              word_count     at  0*RP.storage_units_per_word range  0..15;
              source         at  1*RP.storage_units_per_word range  0.. 3;
              destination    at  1*RP.storage_units_per_word range  4.. 7;
              message_number at  1*RP.storage_units_per_word range  8..15;
              effective_time at  2*RP.storage_units_per_word range  0..31;
--"B"         init_quat_0    at  4*RP.storage_units_per_word range  0..15;
              init_quat_0    at  4*RP.storage_units_per_word range  0..15; --"A"
              init_quat_1    at  5*RP.storage_units_per_word range  0..15; --"A"
--"B"         init_quat_1    at  5*RP.storage_units_per_word range  0..15;
          end record;

--VAX   for initat_messages'size use BI.message_size;
--"B"   for initat_messages'size use BI.message_size+32;
        for initat_messages'size use BI.message_size; --"A"
```

**Figure 11.** Code for Three Compilers Combined in One File

CMS). CMS was also used for configuration control of the Top-Level Design Document, the Test Plan, the Test Procedure, and the Final Technical Report.

### (7) Management Tools

A development status database, implemented using ORACLE (a commercially available relational database), helped track the status and size of the 11th Missile software. Each developer was responsible for updating the database as appropriate. A program automatically generated a weekly status report and sent it to all the developers and management.

The "smart" code counter (see Volume I, section II.2.e(5)) was used to count the individual modules for the software size fields of the database. An ORACLE command file calculated the total software size.

# SECTION III

# EVALUATION OF THE CAMP PARTS
# AND
# THEIR USE IN THE 11TH MISSILE APPLICATION

Two versions of the 11th Missile Application were designed and tested. The first used the CAMP parts, and is referred to as the "Parts Method". This was a complete implementation of the 11th Missile requirements. The second implementation used the CAMP parts and parts composition system (PCS) and is covered in Section IV; it is referred to as the "PCS Method".

## 1. PRODUCTIVITY

During the 11th Missile Application development, effort data was maintained in order to determine the effect of CAMP parts and PCS usage on productivity. In analyzing this data, one very important issue was highlighted: Use of Ada for RTE applications requires mature, highly optimized compilers. Although great strides have been made in the development of Ada compilers in general — the DEC VAX compiler is a good example of this — Ada cross-compilers for RTE applications are not yet fully mature. This situation is similar to the one that existed several years ago when high-quality Ada compilers for non-RTE targets were hard to come by. This is characteristic of the cyclic development of technology in general. As the demand and need for higher quality and greater efficiency in Ada cross-compilers increases, compiler developers will find increasing incentive to expend the resources needed to improve their products and incorporate the features that the 11th Missile team found to be so important for developing effective RTE applications.

With this in mind, it is not surprising that the 11th Missile development team spent a significant amount of time debugging the 1750A cross-compiler. The CAMP data indicates that with a mature compiler, a productivity improvement of up to 15% was possible using the CAMP parts. This figure is based on the adjusted testing hours shown in Table 7. Adjustments were based on two factors.

1. There was a disproportionately large test effort due to the immaturity of the Ada/1750A compiler. Of the 153 errors discovered during testing, 96 were compiler errors and 57 were errors in the 11th Missile code or the CAMP parts. Based on this, it seems reasonable to assume that half the test time was spent debugging the compiler.

2. Again, because the Ada/1750A cross-compiler could not generate correct Ada code for portions of the Navigation CSC, and because adequate work-arounds for the problems did not exist, testing of the Navigation CSC was not completed (see Section VI). At the end of the project, it was estimated that another 240 hours would be required to fully test this CSC once the compiler generated correct code.

Therefore, the adjusted test effort estimate is:

$Test\_Effort = 0.5 \times 4779\ hr + 240\ hr = 2630\ hr$

As a cross-check, this is 45% of the adjusted total effort, which is roughly in line with the 40%-design/20%-code/40%-test rule of thumb.

The raw data, without the adjustment for compiler immaturity (i.e., if all of the time spent debugging the compiler is carried as a cost of using the parts), would actually indicate a productivity decrease of 18%. One of the most obvious areas of the compiler's immaturity was in its inability to handle the complex, though perfectly standard, Ada generics used extensively in the CAMP parts.

TABLE 7. 11TH MISSILE EFFORT

| Phase | Effort (hours) Actual | Adjusted |
|---|---|---|
| Requirements | 708 | 708 |
| Architectural Design | 883 | 883 |
| Det. Design & Code | 1306 | 1306 |
| Testing | 4779 | 2630 |
| Other | 371 | 371 |
| Total | 8047 | 5898 |

Table 8 shows the size of the 11th Missile software in both lines-of-code (LOC) and Ada statements. The "generated" code comprises two sparse matrix operators that were generated using portions of preliminary versions of the CAMP parts composition system Kalman Filter Constructor. Parts statement counts are estimated from the overall ratio of statements to LOC for the parts (0.634 statements/LOC). The "other reused" test software is FORTRAN and Harris assembler code used for the hardware-in-the-loop tests. It was estimated that there were 0.9 statements/LOC for this software. Table 9 shows the actual productivity.

TABLE 8. 11TH MISSILE SIZE - PARTS METHOD

| | Lines of Code | State-ments |
|---|---|---|
| Operational Code | | |
| New | 15708 | 8697 |
| Generated | 1108 | 471 |
| Mod. Parts | 897 | 458 |
| Parts | 3911 | 2480* |
| Total | 21624 | 12106 |
| Test Software | | |
| New | 19752 | 12605 |
| Parts | 1114 | 706* |
| Other Reused | 14544 | 13090* |
| Total | 35410 | 26401 |

* Estimated

TABLE 9. 11TH MISSILE PRODUCTIVITY - PARTS METHOD

| | New Operational | All Operational | New Developed | All Developed |
|---|---|---|---|---|
| LOC/Work-Month | 304.5 | 419.2 | 687.4 | 1105.7 |
| Stmt/Work-Month | 168.5 | 234.7 | 413.0 | 746.5 |
| Work-Hours/LOC | 0.51 | 0.37 | 0.23 | 0.14 |
| Work-Hours/Stmt | 0.93 | 0.66 | 0.38 | 0.21 |

CAMP parts constituted 18.1% of the Parts Method implementation of the 11th Missile code and 3.1% of the test code (see Table 8). Using the adjusted effort as a basis, the 11th Missile developers saved 896 work-hours, 15% of the development effort, by using the CAMP parts. Table 10 compares the adjusted effort with the estimated effort required had the parts not been used.

TABLE 10. EFFECT OF PARTS ON 11TH MISSILE EFFORT

| | Effort (hours) | |
|---|---|---|
| Phase | Adjusted With Parts | Estimated Without Parts |
| Requirements | 708 | 708 |
| Architectural Design | 883 | 883 |
| Det. Design & Code | 1306 | 1604 |
| Testing | 2630 | 3228 |
| Other | 371 | 371 |
| Total | 5898 | 6794 |

Effort Saved: 896 hours

Productivity Improvement: 15%

The increased effort that would have been required for the detailed design and coding phase had the parts not been used was estimated as follows:

$$DD\_Rate = \frac{DD\_Effort - DD\_Gen\_Effort}{New\_Code + Mod\_Code}$$

$$DD\_Rate = \frac{1306\ hr - 40\ hr}{15708\ LOC + 897\ LOC}$$

$$DD\_Rate = 0.0762\ hr/LOC$$

$$DD\_Saved = DD\_Rate \times Parts\_Code$$

$$DD\_Saved = 0.0762\ hr/LOC \times 3911\ LOC$$

$$DD\_Saved = 298\ hr$$

where DD_Rate = Detail design and code productivity, hours per line-of-code
DD_Effort = Total detail design and code effort, hours
DD_Gen_Effort = Estimated effort to generate code, hours
New_Code = New operational code, lines-of-code
Mod_Code = Modified parts code, lines-of-code

Parts_Code = Parts code, lines-of-code
DD_Saved = Detail design and code effort saved, hours

This estimate is conservative because it assumes that it took as long to code the modified parts as it did to design and code the new software.

A similar calculation for the increased effort that would have been required to test without the parts yields:

$$Test\_Rate = \frac{Test\_Effort}{New\_Code + Mod\_Code + Gen\_Code}$$

$$Test\_Rate = \frac{2630\ hr}{15708\ LOC + 897\ LOC + 1108\ LOC}$$

$$Test\_Rate = 0.148\ hr/LOC$$

$$Test\_Saved\_Op = Test\_Rate \times Parts\_Code$$

$$Test\_Saved\_Op = 0.148\ hr/LOC \times 3911\ LOC$$

$$Test\_Saved\_Op = 579\ hr$$

where Test_Rate = Test productivity, hours per line-of-code
Test_Effort = Total test effort, hours
Gen_Code = Generated code, lines-of-code
Test_Saved_Op = Test effort saved due to parts in operational code, hours

However, part of the test code was itself parts. Adjust for this as follows:

$$Test\_Saved = \frac{Test\_Code}{New\_Test\_Code + Reused\_Test\_Code} \times Test\_Saved\_Op$$

$$Test\_Saved = \frac{35410\ LOC}{19752\ LOC + 14544\ LOC} \times 579\ hr$$

$$Test\_Saved = 598\ hr$$

where Test_Saved = Test effort saved, hours
New_Test_Code = New test code, lines-of-code
Reused_Test_Code = Reused test code, lines-of-code
Test_Code = Total test code, lines-of-code

New, modified, and generated code are included in the testing "rate base" because all of this code was unit tested; the parts code was not. There is a small error in this calculation because part of the test effort was devoted to debugging parts. The magnitude of the error isn't known, but is certainly less than 100 hours.

The actual development effort was 8047 hours. It may be argued that the time spent debugging the compiler was a cost of using the parts:

- If the CAMP parts had not been used, complex generics would not have been used;

- If complex generics had not been used, fewer compiler problems would have been encountered;

- If fewer compiler problems had been encountered, total effort would have declined.

It is difficult to assess just how much additional test time was due to compiler problems, and how much of that additional time would not have been required if the parts were not used. If all of the estimated

compiler test time is charged as a cost of using the parts, then there as an 18% decrease in productivity, computed as follows:

$$Productivity\_Improvement = \frac{Estimated\_Effort\_Without\_Parts}{Actual\_Effort\_With\_Parts} - 1.0$$

$$Productivity\_Improvement = \frac{6794}{8047} - 1.0$$

$$Productivity\_Improvement = -0.18$$

It is clear that an immature compiler will negate the benefit of using the CAMP parts "as is".

The above discussion does not address the other costs of using parts. For example, some cost is incurred in the requirements and design phases since the software designers must identify the parts that may be used. Data was not kept on these costs during the 11th Missile Application development since the 11th Missile developers were somewhat familiar with the parts and were in close proximity to the parts development group. Some researchers have postulated the cost of reusing parts to be in the range of 4% to 10%.

## 2. PARTS: WHERE THEY WERE USED

CAMP parts comprised about one-fifth of the flight software (see Table 8). However, the percentage of parts varied widely between the LLCSCs (see Figures 12 and 13). These figures clearly show that the interface to the "outside world" is a major area for which there are few CAMP parts. The Message_Processing and Internal_Bus LLCSCs of both processors are large and composed almost entirely of new code. Message_Processing converts data to or from a format compatible with a MIL-STD-1553B bus. Internal_Bus controls the hardware interface to the bus.

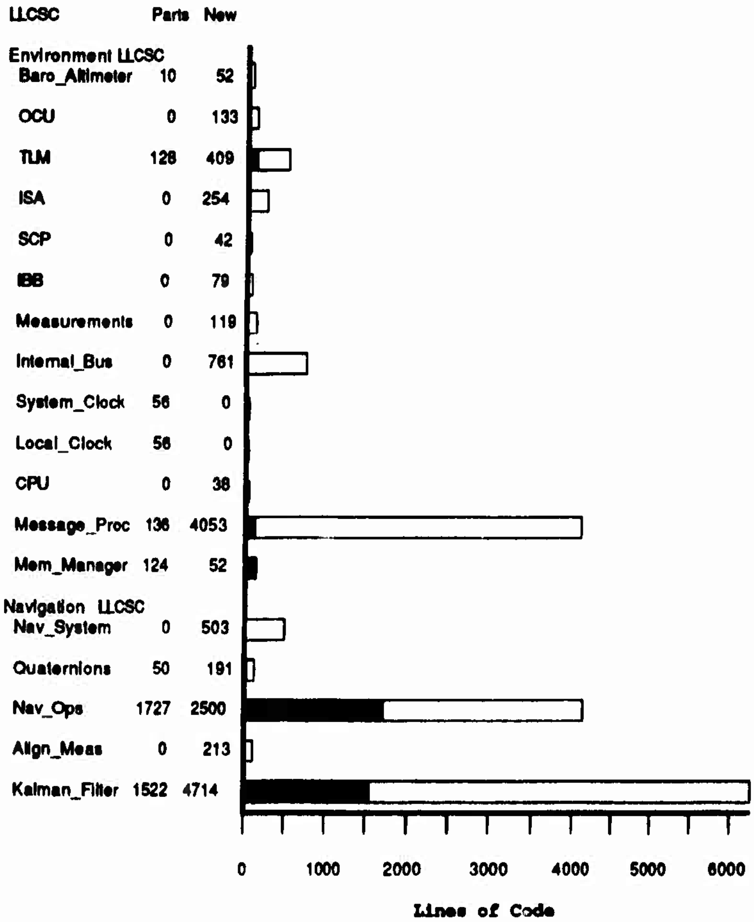


**Figure 12.** NAV Computer Parts Usage

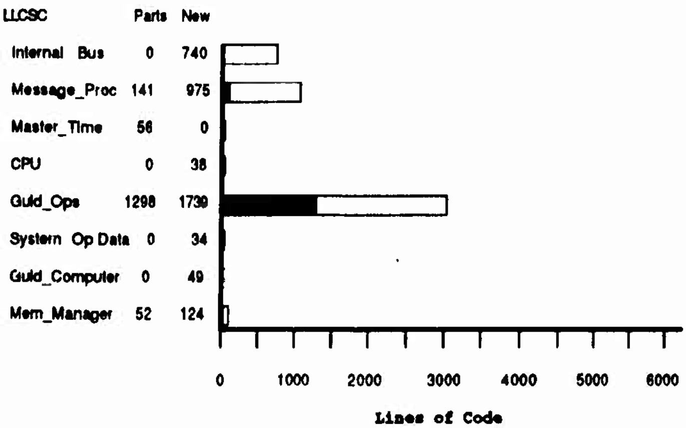


**Figure 13.** GUID Computer Parts Usage

The applicable parts were primarily designed for navigation, guidance, and Kalman filtering. The figures show that most of the parts code is indeed concentrated in the Navigation_Operations, Kalman_ Filter, and Guidance_Operations LLCSCs; however, parts comprise less than half of each of these LLCSCs. New code in these three LLCSCs was needed primarily to provide type definitions and operators, sequence calls to basic functions, and perform functions for which parts are not available or in a manner different from the available parts. New (non-parts) Navigation_Operations code was developed to perform the following functions:

- Define types and operators between them. Most of the operators are scalar multipliers and dividers (e.g., feet / sec => feet per second). This code is a modification of the Basic_Data_Types part, but since most of it was written by the 11th Missile team, it is counted as new code.
- Sequence calls to the navigation functions. The parts provide basic navigation operations (e.g., compute Coriolis acceleration, update velocity), but do not provide a higher level procedure to call them.
- Compute a bias altitude for the barometric altimeter
- Incorporate Kalman corrections
- Implement a third-order barometric-altimeter filter
- Multiply two coordinate-transform matrices and transpose the result

The new Kalman_Filter code was developed to perform the following functions:

- Define types and operators between them. Most of the operators are sparse-matrix operators, some of which are quite large.
- Sequence calls to the Kalman filter functions. The CAMP Kalman_Update part did not meet 11th Missile requirement to handle more than one type of measurement.
- Initialize and integrate the system description (F) matrix.
- Initialize the covariance (P) matrix.
- Define the system noise (Q*) matrix.
- Perform measurement reasonableness tests.
- Compute measurement sensitivity (H) matrices.
- Compute Kalman corrections for the navigator and for other subsystems of the 11th Missile, and maintain a running sum of the corrections.
- Compute whether or not an alignment maneuver is needed, and whether or not the navigator is aligned.
- Provide interfaces to other subsystems.

The new Guidance_Operations code performed the following functions:

- Define types and operators between them. As with Navigation_Operations, most of the operators are scalar multipliers and dividers. This code is another rewrite of Basic_Data_Types.
- Sequence calls to the guidance functions. Once again, the parts provided the basic operations, but did not provide a sequencing procedure.
- Implement a modified first-order filter. The modification allows the user to specify the initial "past values" stored in the filter.
- Implement a lateral-directional "autopilot" that drives a pilot's display instead of rudder and ailerons.

## 3. PARTS: USED, MODIFIED, UNUSED, AND WHY

By the end of testing, the 11th Missile software existed in two versions: one considered the baseline and the other a work-around necessitated by compiler problems (see Section VI). The "baseline" version was the software as it was designed and implemented, assuming a trouble-free compiler. The second or "tested" version was the baseline version after making the many modifications required to work around compiler problems.

The tested version did not differ functionally from the baseline version. However, many times, a compiler deficiency forced the 11th Missile team to search for alternate means of accomplishing a given function. This allowed testing to proceed despite compiler problems.

The following sections explain the CAMP parts usage in both the baseline and tested versions of the 11th Missile Application.

### a. Baseline Version

The 11th Missile baseline flight software used 112 of the 453 parts (24.7%); 96 directly and 16 with modification (see Table 11). A complete list of the parts and how they were used is included in Appendix A. Most of the CAMP parts were used in three LLCSCs: Navigation_Operations, Guidance_Operations, and Kalman_Filter (see Table 12). The following data pertains to the use of CAMP parts in the baseline code.

TABLE 11. SUMMARY OF CAMP PARTS USAGE - PARTS METHOD

| | |
|---|---|
| Parts Used | 96 |
| Parts Used with Modification | 16 |
| Parts Not used | 341 |
| | 453 |

Used 24.7% of Parts

Used 23.1% of Parts Lines-of-Code

#### (1) Parts Modified

Table 13 lists the sixteen parts that were modified. The reasons for modifying the parts are discussed below.

- Basic_Data_Types: Although it can be compiled, it is not intended to be used "as is". It contains only the types and operators needed to instantiate the navigation parts and it declares all floating-point types System.Max_Digits. Any application will need to define additional types and operators and specify the actual precision desired for each type. The 11th Missile team created two versions of Basic_Data_Types, one for Nav (Nav_Computer_Data_Types) and one for Guid (Guidance_Data_Types).

TABLE 12. SUMMARY OF CAMP PARTS USED

| Part | Where Used | Number Used |
|---|---|---|
| Basic Data Types | Navigation_Operations<br>Guidance_Operations | 1 |
| Abstract Processes | All tasks | 1 |
| Standard Trig and Polynomials | Navigation_Operations<br>Guidance_Operations | 22 |
| Coordinate Vector Matrix Algebra and General Purpose Math | Navigation_Operations<br>Guidance_Operations | 11 |
| WGS72 Ellipsoid | Navigation_Operations<br>Guidance_Operations | 3 |
| Common & Wander-Azimuth Navigation | Navigation_Operations | 14 |
| Direction Cosine Matrix Operations | Navigation_Operations | 14 |
| General Vector-Matrix Algebra | Kalman_Filter<br>Navigation_Operations | 11 |
| Kalman Filter | Kalman_Filter | 8 |
| Abstract Data Structures | Memory_Manager (N&G)<br>Kalman_Filter | 3 |
| Quaternion_Operations | Quaternions | 2 |
| Waypoint Steering and Geometric Operations | Guidance_Operations | 8 |
| Signal Processing | Guidance_Operations | 5 |
| Clock Handler | System_Time (N&G)<br>Local_Time | 1 |
| Universal Constants | Navigation_Operations<br>Guidance_Operations<br>Message_Manager (N&G) | 1 |
| Conversion Factors and Unit Conversions | Message_Manager (N&G) | 5 |
| Bus Interface | BIM_Interface (N&G) | 1 |
| External Form Conversion | Barometric_Altimeter | 1 |
| | | 112 |

"Where Used" is the LLCSC which instantiates or imports the part.

TABLE 13. CAMP PARTS MODIFIED

```
Basic_Data_Types
WGS72_Ellipsoid_Metric_Data
WGS72_Ellipsoid_Engineering_Data
Standard_Trig
  Sin
  Cos
  Sin_Cos
  Tan
  Arcsin
  Arccos
  Arcsin_Arccos
  Arctan
General_Purpose_Math
  Square_Root
Signal_Processing
  First_Order_Filter
Waypoint_Steering
  Crosstrack_And_Heading_Error_Operations
Bus_Interface_Parts
Abstract_Processes
```

- WGS72: These parts import Basic_Data_Types. The metric part was changed to import Guidance_Data_Types and the engineering (English) part was changed to import Nav_Computer_Data_Types. Constants not used were deleted.

- Standard_Trig: This is another example of a part not meant to be used "as is". The version supplied with the CAMP parts invokes the VAX Math Library. Any non-VAX application will have to rewrite the entire package body (except for function Arctan2) to invoke trigonometric functions appropriate for that application. These functions will normally be Polynomial parts. In addition, most applications will need to invoke Reduction_Operations to map the input to the domain supported by the selected polynomials. Most sine polynomials, for example, are accurate over the domain [-0.5pi <= x <= 0.5pi]. If the input is outside that domain, an input within it that has the same sine must be computed. The 11th Missile Application required two versions of Standard_Trig, one for single-precision floating-point and the other for extended-precision (see Table 14).

- General_Purpose_Math: The Sqrt function invokes the VAX Math Library. It was changed to use the modified Newton-Raphson square-root function from the Polynomials package.

- Signal_Processing: First_Order_Filter was modified so that the initial past values stored in the package body could be specified. This is required to avoid a control transient when the autopilot is initialized.

- Waypoint_Steering: Crosstrack_And_Heading_Error_Operations was modified to use two-parameter arctangents instead of single-parameters ones. This was required to avoid floating-point overflow when heading error was near $\pm 90$ degrees. First_Order_Filter and Crosstrack_And_Heading_Error_Operations are the only parts that were designed to be used "as is" that had to be modified.

- Bus_Interface_Parts and Abstract_Processes: These are schematic parts, i.e.,they serve as a guide to the implementor (see Volume I, Section I.2.a).

**TABLE 14. POLYNOMIAL PARTS USED FOR TRIGONOMETRIC FUNCTIONS**

| | Polynomial | |
|---|---|---|
| Function | Single-Precision | Extended-Precision |
| Sine | Hastings.Sin_R_4term | Hastings.Sin_R_5term |
| Cosine | Hastings.Cos_R_4term | Hastings.Cos_R_5term |
| Tangent | Hastings.Tan_R_4term | Hastings.Tan_R_5term |
| Arctangent | Hastings.Arctan_R_6term | Hastings.Mod_Arctan_R_8term |
| Arcsine | Fike.Arcsin_S_6term | Fike.Arcsin_S_6term |
| Arccosine | Fike.Arccos_S_6term | Fike.Arccos_S_6term |

### (2) Parts Not Used

Over three-quarters of the CAMP parts were not used by the 11th Missile Application (see Table 11). The primary reasons are (1) the 11th Missile Application did not implement all the functions for which parts are designed, and (2) the parts provide duplicate implementations of some functions both in the baseline and tested versions (see Table 15). A complete list of the parts not used is part of Appendix A.

TABLE 15. SUMMARY OF PARTS NOT USED BY 11TH MISSILE

| | |
|---|---|
| 142 | Not applicable |
| 179 | Duplicate |
| 20 | Incompatible |
| 341 | |

A part is "not applicable" if it implements a function not required by the 11th Missile (e.g., logarithm, Radar_Altimeter). A part is "duplicate" if it implements the same function as a part that is used by the 11th Missile. For example, there are many parts that compute the sine of an angle. All sine parts not used were counted as duplicates. A part is "incompatible" if it performs a function required by the 11th Missile, but was not used. The incompatible parts are further discussed below and are summarized in Table 16.

The 11th Missile team chose statically sparse representations of some Kalman filter matrices. This required writing matrix operations (e.g., multipliers, Set_To_Identity) tailored to the representations. The first eleven of the General_Vector_Matrix_Algebra (GVMA) parts listed in Table 16 are incompatible because they duplicate the function of these statically-sparse-matrix operators. The remaining two GVMA parts (both named Change_Element) were not used because they could be replaced by simple assignment statements. They would have been used if the Ada/1750A compiler used by the 11th Missile team had implemented PRAGMA Inline. The Coordinate_Vector_Matrix_Algebra Set_To_Zero_Vector function was not used for the same reason.

The five Kalman_Filter parts listed were not used because they were limited to handling a single measurement type (i.e., a single type of measurement sensitivity matrix). The 11th Missile Application had to handle four types of measurements.

Lateral_Directional_Autopilot was not used because it generates rudder and aileron commands, and the 11th Missile Application was not designed to fly a missile; instead it drives a roll error needle for the pilot of a test aircraft.

TABLE 16. PARTS INCOMPATIBLE WITH 11TH MISSILE

```
General_Vector_Matrix_Algebra
  ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix
  ABA_Trans_Vector_Sq_Matrix
  ABA_Trans_Vector_Scalar
  Dot_Product_Operations_Restricted
  Matrix_Vector_Multiply_Unrestricted
  Matrix_Vector_Multiply_Restricted
  Vector_Matrix_Multiply_Restricted
  Dynamically_Sparse_Matrix_Operations_Unconstrained
    Set_To_Zero_Matrix
    Add_To_Identity
    Subtract_From_Identity
    Set_To_Identity_Matrix
  Symmetric_Full_Storage_Matrix_Operations_Constrained
    Change_Element
  Diagonal_Matrix_Operations
    Change_Element

Coordinate_Vector_Matrix_Algebra
  Set_To_Zero_Vector

Kalman_Filter_Data_Types

Kalman_Filter_Compact_H_Parts
  Sequentially_Update_Covariance_Matrix_And_State_Vector
  Kalman_Update

Kalman_Filter_Complicated_H_Parts
  Sequentially_Update_Covariance_Matrix_And_State_Vector
  Kalman_Update

Autopilot
  Lateral_Directional_Autopilot
```

## b. Tested Version

The set of CAMP parts used in the tested version of the flight software was identical to the set used in the baseline version. However, due to compiler difficulties (see Section VI), it was necessary to make further modifications in the tested version . Most of the changes involved using the CAMP generic code as templates for constructing "manual instantiations", i.e., converting the part to an equivalent non-generic version. Manual instantiations were necessary whenever the compiler failed in compilation or produced incorrect, malfunctioning code for a generic unit. The process was carried out by editing CAMP code via batch editing commands and inserting the modified code into the software in place of associated instantiations.

Virtually all of the generic CAMP parts in the Guid_Computer software were replaced with manual instantiations. In the Nav_Computer, a smaller proportion of generic units required this treatment. Tables 17 and 18 list the CAMP parts which were modified by manual instantiation.

Minor additional changes were made to the CAMP Kalman filter parts because of the need to reduce the use of run-time stack storage. The Propagate routine in the Error_Covariance_Matrix_Manager of the Kalman_Filter_Common_Parts contained an expression which proved to quite inefficient. A similar problem existed in the Update_Error_Covariance_Matrix_General_Form routines of the

TABLE 17. PARTS MANUALLY INSTANTIATED IN GUID_COMPUTER

```
Abstract Data Structures
  Bounded FIFO Buffer
  Nonblocking Circular Buffer

Coordinate Vector Matrix Algebra
  Cross Product
  Vector Operations
  Vector Scalar Operations

Polynomials
  Fike Semicircle Operations
  Hastings Radian Operations
  Modified Newton Raphson Square Root
  Reduction Operations

Signal Processing Parts
  Absolute Limiter
  Lower Limiter
  Upper Lower Limiter

Waypoint Steering
  Compute Turn Angle And Direction
  Compute Turning And Nonturning Distances
  Crosstrack And Heading Error Operations
  Distance to Current Waypoint With Arcsin
  Steering Vector Operations
  Turn Test Operations
```

Kalman_Filter_Compact and Kalman_Filter_Complicated packages. The expressions in these units contained operations over large matrix types. Unfortunately, the compiler's allocation of temporary space for operator results was rather primitive and very inefficient. In order to improve efficiency, the troublesome expressions had to be split up and evaluated over several assignment statements. This is discussed further in Section VI.

## 4. PARTS CHANGED

Twenty-three Software Change Proposals or Software Enhancement Proposals were written for the CAMP parts as a result of the 11th Missile development (see Table 19). These resulted in 62 changes to 53 different CAMP parts. The changes consisted of 20 additions, 34 enhancements, and 8 "fixes".

TABLE 18. PARTS MANUALLY INSTANTIATED IN NAV_COMPUTER

```
Abstract Data Structures
  Bounded FIFO Buffer
  Nonblocking Circular Buffer
  Unbounded Priority Queue

Common Navigation Parts
  Update Velocity

Wander Azimuth Navigation Parts
  Compute Coriolis Acceleration

General Vector Matrix Algebra
  Column Matrix Operations
    ABA Symm Transpose
  Diagonal Full Matrix Add Restricted
  Diagonal Matrix Operations
  Diagonal Matrix Scalar Operations
  Matrix Matrix Multiply Restricted
  Symmetric Full Storage Matrix Operations Constrained
  Vector Vector Transpose Multiply Restricted
  Vector Scalar Operations Constrained

Kalman Filter Common Parts
  Error Covariance Matrix Manager
  State Transition And Process Noise Matrices Manager

Kalman Filter Compact H Parts
  Compute Kalman Gain
  Update Error Covariance Matrix General Form
  Update State Vector

Kalman Filter Complicated H Parts
  Compute Kalman Gain
  Update Error Covariance Matrix General Form
  Update State Vector
```

## TABLE 19. PARTS CHANGES AND ENHANCEMENTS GENERATED BY THE 11TH MISSILE DEVELOPMENT

(Part 1 of 4)

| SCP/SEP | Description | Part(s) Affected | Class |
|---|---|---|---|
| C | Modify FIFO so it can be used to pass information between tasks | Abstract_Data_Structures | |
| | | Bounded_FIFO_Buffer | E |
| D | Add parts that do not use small angle approximation to compute distance | Waypoint_Steering | |
| | | Steering_Vector_Operations_with_Arcsin | A |
| | | Distance_To_Current_Waypoint_With_Arcsin | A |
| | | Geometric_Operations | |
| | | Compute_Segment_And_Unit_Normal_Vector_With-_Arcsin | A |
| F | Add two-parameter arctangent | Standard_Trig | |
| | | Arctan2 | A |
| 002 | Add operators to support Kalman_Filter revisions | General_Vector_Matrix_Algebra | |
| | | Column_Matrix_Operations | |
| | | Set_Diagonal_and_Subtract_From_Identity | A |
| | | ABA_Transpose | A |
| | | ABA_Symm_Transpose | A |
| | | ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix | |
| | | ABA_Transpose | A |
| | | ABA_Trans_Vector_Sq_Matrix | |
| | | ABA_Transpose | A |
| | | ABA_Trans_Vector_Scalar | |
| | | ABA_Transpose | A |
| 003 | Add general form of Update_P | Kalman_Filter_Compact_H_Parts | |
| | | Update_Error_Covariance_Matrix_General_Form | A |
| | | Kalman_Filter_Complicated_H_Parts | |
| | | Update_Error_Covariance_Matrix_General_Form | A |
| 004 | Revise and extend number of types in Kalman Filter parts | Kalman_Filter_Common_Parts | |
| | | State_Transition_and_Process_Noise_Matrices-_Manager | E |
| | | Error_Covariance_Matrix_Manager | E |
| | | State_Transition_Matrix_Manager | E |

Class Key: A - New part added
E - Part enhanced
F - Part fixed

TABLE 19. PARTS CHANGES AND ENHANCEMENTS GENERATED BY THE 11TH MISSILE DEVELOPMENT

(Part 2 of 4)

| SCP/SEP | Description | Part(s) Affected | Class |
|---|---|---|---|
| 004 (cont) | | Kalman_Filter_Compact_H_Parts | |
| | | Compute_Kalman_Gain | E |
| | | Update_Error_Covariance_Matrix | E |
| | | Update_State_Vector | E |
| | | Sequentially_Update_Covariance_Matrix_and-State_Vector | E |
| | | Kalman_Update | E |
| | | Update_Error_Covariance_Matrix_General_Form | E |
| | | Kalman_Filter_Complicated_H_Parts | |
| | | Compute_Kalman_Gain | E |
| | | Update_Error_Covariance_Matrix | E |
| | | Update_State_Vector | E |
| | | Sequentially_Update_Covariance_Matrix_and-State_Vector | E |
| | | Kalman_Update | E |
| | | Update_Error_Covariance_Matrix_General_Form | E |
| 006 | Extend number of types | Waypoint_Steering | |
| | | Steering_Vector_Operations | E |
| | | Steering_Vector_Operations_with_Arcsin | E |
| | | Crosstrack_and_Heading_Error_Operations | E |
| | | Distance_To_Current_Waypoint | E |
| | | Distance_To_Current_Waypoint_with_Arcsin | E |
| | | Geometric_Operations | |
| | | Compute_Segment_And_Unit_Normal_Vector | E |
| | | Compute_Segment_And_Unit_Normal_Vector_with-Arcsin | E |
| | | Great_Circle_Arc_Length | E |
| 008 | Add capability to vary gravitational acceleration with altitude | Common_Navigation_Parts | |
| | | Update_Velocity | E |
| 009 | Add reset functions to filters | Signal_Processing | |
| | | General_First_Order_Filter | E |
| | | Tustin_Lead_Lag_Filter | E |
| | | Tustin_Lag_Filter | E |
| | | Second_Order_Filter | E |

Class Key: A - New part added
E - Part enhanced
F - Part fixed

## TABLE 19. PARTS CHANGES AND ENHANCEMENTS GENERATED BY THE 11TH MISSILE DEVELOPMENT

(Part 3 of 4)

| SCP/SEP | Description | Part(s) Affected | Class |
|---|---|---|---|
| 016 | Unfold loops to decrease execution time | Direction_Cosine_Matrix_Operations<br>DCM_General_Operations<br>Perform_Rectangular_Integration_of_DCM | <br><br>E |
| 017 | Correct rotation rate equations | Wander_Azimuth_Navigation_Parts<br>Compute_Earth_Relative_Navigation_Rotation_Rate | <br>F |
| 018 | Use Arctan2 to compute heading | Common_Navigation_Parts<br>Compute_Heading | <br>F |
| 019 | Fix north velocity computation | Wander_Azimuth_Navigation_Parts<br>Compute_North_Velocity<br>Compute_Earth_Relative_Horizontal_Velocities | <br>F<br>F |
| 020 | Compute latitude, longitude, and wander angle using two-parameter arctangent | Wander_Azimuth_Navigation_Parts<br>Compute_Latitude_Using_Two_Value_Arctangent<br>Compute_Longitude_Using_Two_Value_Arctangent<br>Compute_Wander_Azimuth_Angle_Using_Two_Value-_Arctangent | <br>A<br>A<br>A |
| 021 | Add routine that takes sin & cos of wander azimuth as inputs, instead of computing them | Wander_Azimuth_Navigation_Parts<br>Compute_East_Velocity_With_Sin_Cos_In<br>Compute_North_Velocity_With_Sin_Cos_In<br>Compute_Earth_Relative_Horizontal_Velocities-_With_Sin_Cos_In | <br>A<br>A<br>A |
| 022 | Gravity term has incorrect sign | Common_Navigation_Parts<br>Update_Velocity | <br>F |
| 023 | Constraint error in square root function when instantiated with a limited-range type | Polynomials<br>Modified_Newton_Raphson<br>SqRt<br>Newton_Raphson<br>SqRt | <br><br>E<br><br>E |

Class Key: A - New part added
E - Part enhanced
F - Part fixed

## TABLE 19. PARTS CHANGES AND ENHANCEMENTS GENERATED BY THE 11TH MISSILE DEVELOPMENT

(Concluded)

| SCP/SEP | Description | Part(s) Affected | Class |
|---|---|---|---|
| 024 | Add parts to normalize input to trig functions | Polynomials<br>Reduction_Operations<br>Sine_Reduction<br>Cosine_Reduction | <br><br>A<br>A |
| 025 | Modify GPMath Square Root function to be able to instantiate the revised Modified_Newton_Raphson.SqRt | General_Purpose_Math<br>Square_Root | <br>E |
| 026 | Two-parameter arctangent inaccurate near +/- 90 degrees | Standard_Trig<br>Arctan2 | <br>F |
| 032 | Add Current_Velocity function to Update_Velocity | Common_Navigation_Parts<br>Update_Velocity | <br>E |
| 033 | Correct error in Propagate Q | Kalman_Filter_Common_Parts<br>State_Transition_and_Process_Noise_Matrices-_Manager | <br>F |
| 036 | Compute curvature instead of radius of curvature to avoid divide-by-zero error | Wander_Azimuth_Navigation_Parts<br>Compute_Curvatures | <br>F |

Class Key: A - New part added
E - Part enhanced
F - Part fixed

# SECTION IV

# EVALUATION OF THE PARTS COMPOSITION SYSTEM AND ITS USE IN THE 11TH MISSILE APPLICATION

The 11th Missile Application team designed and coded two versions of the 11th Missile. The first ("Parts Method") used the CAMP parts; this complete implementation of the 11th Missile requirements was covered in Section III. The second ("PCS Method") used the CAMP parts composition system (i.e, the AMPEE system). This was not a completely new implementation as only the Kalman filter was reimplemented and unit tested.

## 1. PRODUCTIVITY

CAMP data indicates that a productivity improvement of up to 28% is possible using the AMPEE system Kalman Filter Constructor. Since the PCS Method was not a complete implementation and was not integration tested, this is a rough estimate. PCS-generated code and CAMP parts constitute 29.8% of the PCS Method implementation of the 11th Missile code (see Table 20). The estimated productivity improvement uses the estimated cost of developing the software without parts as a basis (see Section III.1 and Table 21).

**TABLE 20. 11TH MISSILE SIZE - PCS METHOD**

| | Lines of Code | Statements |
|---|---|---|
| Operational Code | | |
| New | 14707 | 8206 |
| Generated | 2680 | 987 |
| Mod. Parts | 897 | 458 |
| Parts | 3946 | 2491* |
| Total | 22230 | 12142 |

* Estimated

The reduction in the detailed design and coding phase was estimated as follows:

$$DD\_Saved = DD\_Rate \times (Gen\_Code + Parts\_Code) - PCS\_Cost$$

$$DD\_Saved = 0.0762\ hr/LOC \times (2680\ LOC + 3946\ LOC) - 13\ hr$$

$$DD\_Saved = 492\ hr$$

where DD_Saved = Detail design and code effort saved, hours
DD_Rate = Detail design and code productivity, hours per line-of-code
GenCode = Generated code, lines-of-code
Parts_Code = Parts code, lines-of-code
PCS_Cost = Cost of using PCS, hours

DD_Rate is from Section III.1. The PCS_Cost is the time spent at the PCS generating the Kalman filter, plus the time spent writing the code to interface the generated code to the rest of the Parts Method

**TABLE 21. ESTIMATED EFFECT OF PCS ON 11TH MISSILE EFFORT**

| Phase | Effort (hours) Estimated Without Parts/PCS | Estimated With Parts/PCS |
|---|---|---|
| Requirements | 708 | 708 |
| Architectural Design | 883 | 883 |
| Det. Design & Code | 1604 | 1112 |
| Testing | 3228 | 2247 |
| Other | 371 | 371 |
| Total | 6794 | 5321 |

Effort Saved: 1473 hours

Productivity Improvement: 28%

code (mostly renaming instantiations and types). Thus, the 13 hour cost includes some extra work, because a "from scratch" application would not require the interface code. On the other hand, another development team would not be as familiar with the PCS and the parts as the 11th Missile team; this would drive the cost up.

The test effort that could be saved was estimated similarly:

$$Test\_Saved = Test\_Rate \times (Gen\_Code + Parts\_Code)$$

$$Test\_Saved = 0.148\ hr/LOC \times (2680\ LOC + 3946\ LOC)$$

$$Test\_Saved = 981\ hr$$

where
Test_Saved = Test effort saved, hours
Test_Rate = Test productivity, hours per line-of-code
GenCode = Generated code, lines-of-code
Parts_Code = Parts code, lines-of-code

Test_Rate is from Section III.1. This estimate assumes that the parts and the PCS-generated code would not be unit tested. Under this assumption, none of the test code would have been CAMP parts, so no credit is given for that. Also, as in Section III.1, this estimate ignores the other costs of using the parts and the PCS. Therefore, 28% is a ceiling on the possible improvement in productivity to be gained from using the Kalman Filter Constructor.

## 2. PCS: WHERE IT WAS USED

The Kalman Filter Constructor was the only AMPEE system facility used. The PCS-generated code and the parts instantiated by it constitute 70.1% of the Kalman_Filter LLCSC (compared to 24.4% with the Parts Method). The new (i.e., not parts or PCS-generated) Kalman_Filter code was the same for both methods, with two exceptions:

- The amount of new code that defines types and operators was greatly reduced with the PCS Method. The PCS generated most of the type descriptions, all of the sparse-matrix operators, and all of the operator instantiations.

- The PCS Method required new code to rename procedures written or instantiated by the PCS. This was required to make the PCS-generated code compatible with the rest of the Parts Method implementation.

## 3. PARTS: USED, MODIFIED, UNUSED, AND WHY

The PCS Method used the same set of parts as the Parts Method with one exception (see Table 22). The exception occurred because the PCS used a part that was written after the Parts Method code was designed. The new part allowed the user to instantiate a vector*scalar*vector-transpose operation, instead of having to write one using the vector*vector-transpose part.

TABLE 22. SUMMARY OF CAMP PARTS USAGE - PCS METHOD

| | |
|---|---|
| Parts Used | 96 |
| Parts Used with Modification | 16 |
| Parts Not used | 341 |
| | 453 |

Used 24.7% of Parts
Used 23.3% of Parts Lines-of-Code

## 4. PCS: PROBLEMS

The Kalman Filter Constructor, as currently implemented, has two major options for representation of Kalman matrices: full or sparse. The full-matrix code uses less instruction memory, but more operand memory and is relatively slow. The sparse-matrix code uses much more instruction memory, but is relatively fast. In the case of the 11th Missile Application, the generated sparse-matrix code caused the program to exceed the 64K-word instruction memory limit imposed by the compiler. The full-matrix code, had it been generated, would have been very similar to the Parts Method implementation, which also exceeded the 64K-word operand memory limit imposed by the compiler. As a result, PCS-generated code was not hardware-in-the-loop tested.

One or more options that generate code with intermediate speed and memory usage must be added to the Kalman Filter Constructor. One possibility is to represent F as an array of records, where each record includes a set of matrix indices and the value of the corresponding matrix element (see Figure 14).

Another project at MDAC-STL used the PCS to generate a 17-state Kalman filter for a flight demonstration of a GPS-aided navigation system. The application used a ruggedized MicroVAX with eight megabytes of physical memory. The PCS-generated sparse-matrix code worked well in this application.

```
type kalman_elements is digits 9;

type states is (x_pos,        y_pos,        z_pos,
                x_vel,        y_vel,        z_vel,
                                            z_acc,
                x_att,        y_att,        z_att,
                x_gyro_bias,  y_gyro_bias,  z_gyro_bias,
                x_acc_bias,   y_acc_bias,   z_acc_bias,
                              y_acc_scale,  z_acc_scale,
                prop_1,       prop_2,       prop_3);

type f_elements is
    record
        row   : states;
        col   : states;
        value : kalman_elements;
    end record;

f_size : constant := 76;
type f_indices is new integer range 1 .. f_size;

type f_matrices is array f_indices of f_elements;
```

Figure 14. Possible New Representation of Kalman Matrix

# SECTION V

# EVALUATION OF ADA AND ITS USE IN THE 11TH MISSILE APPLICATION

The CAMP 11th Missile Application served as a proving ground not only for the CAMP parts, but also for the Ada language. This section shows that Ada is effective for real-time embedded applications and that the "optional" features of the language must be implemented.

Ada was shown to be an effective language. The 11th Missile Application required only 21 lines of assembly code — 0.1% of the total application code. No assembly code would have been required if the Start-up ROM software had been designed to interface to the Ada compiler's Run-Time System. This is particularly impressive in view of the low-level machine-interface functions implemented.

## 1. EFFECTIVENESS OF ADA FOR MACHINE INPUT/OUTPUT - AN EXAMPLE

The effectiveness of Ada for low-level machine-interface functions was demonstrated by the fact that the Bus Interface Module (BIM) Interface was coded entirely in Ada.

### a. Description of the Bus Interface Module (BIM) Interface

The Bus Interface Module (BIM) is a hardware device-controller comprising a Motorola 68000 microprocessor and associated circuitry packaged on a single card. It receives and sends messages on a MIL-STD-1553B data bus, places messages into 1750A memory, and receives messages from the 1750A.

A BIM may be operated in one of two modes: Bus Controller (BC) or Remote Terminal (RT). A Bus Controller polls all terminals (including itself) on the bus and sends transmit and receive commands to other terminals as required. The BIM communicates with the 1750A via a command port, interrupts, and direct memory access (DMA). With the exception of the command word, all data transfers are via DMA (see Figure 15).

The 1750A issues commands to the BIM via a bit-mapped 16-bit output register (see Table 23). The register is accessed by a Programmed Output (PO) instruction; its address is 610 (for a RT) or 620 (for a BC). *Command_Status* (see Figure 15) is updated each time the BIM responds to a command, and *Error_Status* is updated whenever there is a problem receiving a message. The two status words are bit-mapped and have the same format (see Table 24). *Index* indexes *Input_PTR*, an array of 16 pointers to input messages. When an input message is received, the BIM increments *Index*, then copies the message to the address specified by *Input_PTR(Index)*. *Output_Ptr* points to an output message. The BIM reads it when commanded to transmit a message, then reads the message. *Polling_List* is an array containing a polling sequence. It is used only by a BC, and only if the polling sequence is being changed.

The 1750A addresses of *Command_Status, Error_Status, Index, Input_Ptr, Output_Ptr*, and *Polling_List*, are among the items specified in a BIM Initialization Block (see Table 25). When the BIM is initialized, the address of the initialization block is sent to the BIM via the command port.

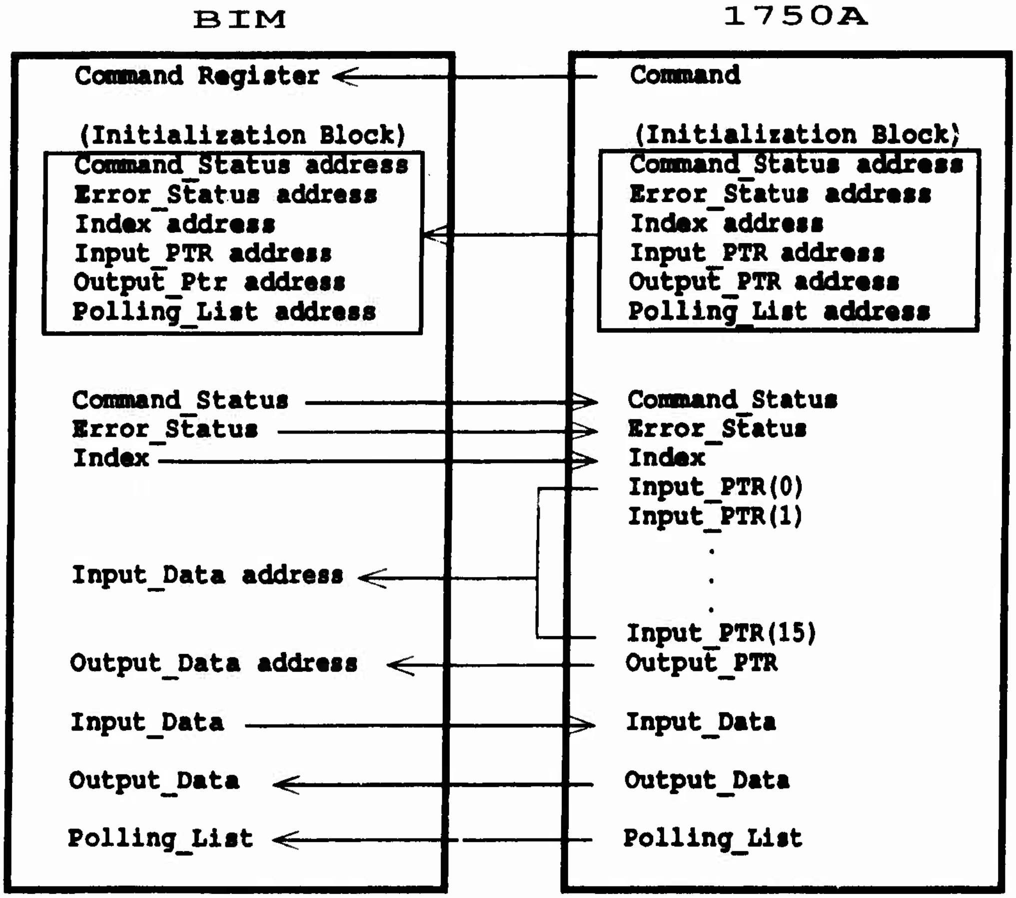


**Figure 15. BIM/1750A Interface**

**TABLE 23. BUS INTERFACE MODULE COMMAND WORD**

| Command Word Format | | |
|---|---|---|
| Bit(s) | Field | Description |
| 00 | OD | 1 => Output Data (send message) |
| 01 | SP | 1 => Stop Polling |
| 02 | CP | 1 => Change Polling Sequence |
| 03 | CM | 1 => Change Number of Message Retries |
| 04 | RP | 1 => Restart Polling |
| 05 | B | 1 => Perform BIT |
| 06-09 | I | Initialize: 0101 for the first command, 1010 for the second |
| 10 | RS | Restart from Timeout |

**TABLE 24. BUS INTERFACE MODULE STATUS WORD**

| Status Word Format | | |
|---|---|---|
| Bit(s) | Field | Description |
| 00 | PA | 1 => Polling Active |
| 01 | MM | Module Mode: 0 => bus controller, 1 => remote terminal |
| 03 | MI | 1 => Module Initialized |
| 04 | CPU | BIT result: 1 => CPU Error |
| 05 | CS | BIT result: 1 => Checksum Error |
| 06 | IM | 1 => Invalid Message |
| 07 | IV | 1 => Invalid Transmit Vector Word |
| 08-12 | RT | RT associated with IM or IV |
| 13 | TI | 1 => Time-out Error |

**TABLE 25. BUS INTERFACE MODULE INITIALIZATION BLOCK**

| BIM Initialization Block Format | | |
|---|---|---|
| Word | Bit(s) | Description |
| 00 | 00-15 | Initialization Block Word Count |
| 01 | 15 | Module Mode: 0 => BC, 1 => RT |
| 02 | 11-15 | Terminal ID |
| 03 | 08-11 | Input_PTR address state |
| 03 | 12-15 | Input_PTR processor state |
| 04 | 00-15 | Input_PTR address |
| 05 | 12-15 | Maximum index value (length of array Input_PTR) |
| 06 | 08-11 | Output_PTR address state |
| 06 | 12-15 | Output_PTR processor state |
| 07 | 00-15 | Output_PTR address |
| 08 | 08-11 | Command_Status address state |
| 08 | 12-15 | Command_Status processor state |
| 09 | 00-15 | Command_Status address |
| 10 | 08-11 | Error_Status address state |
| 10 | 12-15 | Error_Status processor state |
| 11 | 00-15 | Error_Status address |
| 12 | 08-11 | Index address state |
| 12 | 12-15 | Index processor state |
| 13 | 00-15 | Index address |
| 14 | 00-15 | Number of Message Retries (BC only) |
| 15 | 08-11 | Polling_List address state (BC only) |
| 15 | 12-15 | Polling_List processor state (BC only) |
| 16 | 00-15 | Polling_List address (BC only) |
| 17 | 11-15 | Number of valid terminal IDs (BC only) |
| 18 - 18+N-1 | 11-15 | Terminal IDs (N entries, BC only) |
| 18+N - 18+N+M-1 | 00-15 | Polling Sequence (M entries, BC only) |

*Input_Ptr* is the most extreme example of indirect addressing in the BIM interface: triple-nested pointers. The top-level pointer is to the BIM Initialization Block. The block contains a pointer to Input_Ptr (second level), where Input_Ptr is an array that points to the actual message locations (third level).

There are two interrupts for each BIM. The input interrupt (level 11 for BC, 13 for RT) signals that a message has been transferred to 1750A memory. The output interrupt (level 12 for BC, 14 for RT) signals that an output message has been sent over the bus and that the BIM is ready to accept another output message.

### b. Ada Solution to the BIM Interface

The BIM interface presents a significant challenge to a higher-order language programmer. In the past, the use of interrupt handlers, a bit-mapped command port, bit-mapped status words, and pointers (including double- and triple-nested pointers) would have required programming in assembly language. Ada's access types and representation specification features made it possible to program the BIM interface entirely in Ada.

The interrupt handlers were implemented as Ada tasks with interrupt entries (see Figure 16).

```
task Input_Interrupt_Handler is

    PRAGMA Priority (11);

    entry Interrupt;
    for Interrupt use at 13;

end Input_Interrupt_Handler;
```

**Figure 16.** Use of Interrupt Entry

The command port was modeled as a record, with the bit-map defined by a record representation clause (see Figure 17). This code example demonstrates the use of several optional features of Ada: change of representation (short_boolean and memory_states), size clauses for both scalar and record types, enumeration representation (init_commands), record representation, and nested representations (both command_words and its components have representation specs). The example also shows the use of an Ada predefined constant to make the code compiler-independent. Using System.storage_unit ensured that the representation clauses would work for either an 8 or 16 bit storage unit.

The command port was accessed via package Low_Level_IO. Low_Level_IO, which was provided by the compiler vendor, had to be slightly modified because the command port address (610 or 620) is not in the legal range defined by MIL-STD-1750A.

Figure 18 shows how the BIM Initialization Block was coded. This example shows more of the power and flexibility of Ada. The block was coded as a variant record, with a representation clause specifying locations of the components. Not shown are the type definitions of the component types

```
with System;
package Representation_Parameters is

    message_word_size      : constant := 16;  -- bits
    storage_units_per_word : constant := message_word_size/System.storage_unit;

end Representation_Parameters;


package BIM_Interface_Types is

    type short_boolean is new boolean;
    for  short_boolean'size use 1;

end BIM_Interface_Types;


with BIM_Interface_Types;
with Representation_Parameters;
package BIM_Interface_Hidden_Types is

   package BIT renames BIM_Interface_Types;
   package RP  renames Representation_Parameters;

   type init_commands    is (not_init_cmd, init_cmd_1, init_cmd_2);
   for  init_commands use (not_init_cmd => 0,
                           init_cmd_1   => 2#0101#,
                           init_cmd_2   => 2#1010#);
   for  init_commands'size use 4;

   type memory_states is new integer range 0..15;
   for  memory_states'size use 4;

   type command_words is
      record
         output_data              : BIT.short_boolean := BIT.false;
         stop_polling             : BIT.short_boolean := BIT.false;
         change_polling_sequence  : BIT.short_boolean := BIT.false;
         change_message_retries   : BIT.short_boolean := BIT.false;
         restart_polling          : BIT.short_boolean := BIT.false;
         perform_BIM_BIT          : BIT.short_boolean := BIT.false;
         initialize               : init_commands     := not_init_cmd;
         restart_from_timeout     : BIT.short_boolean := BIT.false;
         memory_state             : memory_states     := 0;
      end record;

   for command_words use
      record
         output_data             at 0*RP.storage_units_per_word range  0.. 0;
         stop_polling            at 0*RP.storage_units_per_word range  1.. 1;
         change_polling_sequence at 0*RP.storage_units_per_word range  2.. 2;
         change_message_retries  at 0*RP.storage_units_per_word range  3.. 3;
         restart_polling         at 0*RP.storage_units_per_word range  4.. 4;
         perform_BIM_BIT         at 0*RP.storage_units_per_word range  5.. 5;
         initialize              at 0*RP.storage_units_per_word range  6.. 9;
         restart_from_timeout    at 0*RP.storage_units_per_word range 10..10;
         memory_state            at 0*RP.storage_units_per_word range 12..15;
      end record;

   for command_words'size use 16;

end BIM_Interface_Hidden_Types;
```

Figure 17. Example of Record Representation Clause

within the initialization block; as with the components of Command_Word, these also had representations specified. In addition to representation clauses, access types were used for addresses of data objects.

## 2. INEFFECTIVENESS OF ADA FOR OPERATING SYSTEM INTERFACE

This section examines where assembly language was used in the 11th Missile Application. It will show that Ada was not effective when

- Code must run before the Ada Run-Time System starts, or
- Machine code must be at a specified memory location.

Nineteen of the 21 lines of assembly code in the 11th Missile Application were in module Reset_Sys_Enable_ROM (see Figure 19). This module interfaces with the Start-up Real-time Multi-tasking Operating System (SURMOS). Reset_Sys_Enable_ROM contains both the first code and the last code to be executed from RAM.

SURMOS is a ROM-resident operating system used at power-up to provide basic services: performing built-in tests, BIM initialization and control, telemetry, and downloading the 11th Missile Application software to RAM.

SURMOS was designed to interface with the original Real-time Multi-Tasking Operating System (RMOS)/JOVIAL implementation of the 11th Missile Application. It was used without modification for the Ada implementation. If SURMOS had been designed to interface with the Ada Run-Time System, it probably would not have been necessary to code the 11th Missile interface to it in assembly language. However, as written, SURMOS does not transfer control to the location expected by the Ada Run-Time System, nor does it leave appropriate values in the page registers.

RAM and ROM execute in the same address space. For example, if the instruction counter is 40 (hex), either ROM address 40 or RAM address 40 will be executed, depending on whether ROM is enabled or disabled. At power-up, ROM is enabled. When SURMOS is commanded to start the application software, it executes an instruction at location 3E to disable ROM (see Figure 20). Because the hardware "pre-fetches" the next instruction, it will then execute the instruction at location 40 in ROM, which is a branch to location 40. This means the first instruction of the 11th Missile code to be executed must be located at address 40 (hex) in RAM. The Ada Run-Time System expects control to be transferred to location 0.

SURMOS does not leave the page registers in the power-on reset state, which is a 1-to-1 correspondence between physical and logical memory. Since the Ada Run-Time System expects the page registers in the reset state, the initialization code must set the page registers. This is done in the PR_LOOP (see Figure 19).

Finally, the start-up code loads the initial status (the LST INIT_STATUS instruction). This restores the status (interrupt mask, status word, and instruction counter) to the power-up state and results in a branch to location zero, which transfers control to the Ada Run-Time System code.

Control may be returned to SURMOS at location 44. To do this, the 11th Missile Application must

```
type Initialization_Block
                 (bim_mode                  : module_modes := bus_controller;
                  terminals                 : number_of_terminals := 1) is
   record
      word_count             : word_counts_range;
      terminal_id_msb        : RT_ID_msb_constant := 1;
      terminal_id            : BT.terminals;
      input_ptrs_blk_AS      : memory_states;
      input_ptrs_blk_PS      : memory_states;
      input_ptrs_blk_addr    : input_ptrs_block_ptrs;
      no_of_input_buffers    : index_word;
      output_ptr_AS          : memory_states;
      output_ptr_PS          : memory_states;
      output_ptr_addr        : output_ptrs_addr_type;
      command_status_AS      : memory_states;
      command_status_PS      : memory_states;
      command_status_addr    : status_word_ptrs;
      error_status_AS        : memory_states;
      error_status_PS        : memory_states;
      error_status_addr      : status_word_ptrs;
      input_index_AS         : memory_states;
      input_index_PS         : memory_states;
      input_index_addr       : input_index_addr_type;
      case bim_mode is
         when bus_controller =>
            retries                : retries_type;
            new_poll_retries_AS    : memory_states;
            new_poll_retries_PS    : memory_states;
            new_poll_retries_addr  : new_poll_retries_addr_type
                                          := new new_poll_retries_block;
            terminal_ids_count     : number_of_terminals;
            rt_ids_and_poll_seq    : BIM_RT_and_poll_words_list;
         when others =>
            null;
      end case;
   end record;

for Initialization_Block use
   record
      word_count            at  0*RP.storage_units_per_word range  0 ..   15;
      bim_mode              at  1*RP.storage_units_per_word range 15 ..   15;
      terminal_id_msb       at  2*RP.storage_units_per_word range 11 ..   11;
      terminal_id           at  2*RP.storage_units_per_word range 12 ..   15;
      input_ptrs_blk_AS     at  3*RP.storage_units_per_word range  8 ..   11;
      input_ptrs_blk_PS     at  3*RP.storage_units_per_word range 12 ..   15;
      input_ptrs_blk_addr   at  4*RP.storage_units_per_word range  0 ..   15;
      no_of_input_buffers   at  5*RP.storage_units_per_word range 12 ..   15;
      output_ptr_AS         at  6*RP.storage_units_per_word range  8 ..   11;
      output_ptr_PS         at  6*RP.storage_units_per_word range 12 ..   15;
      output_ptr_addr       at  7*RP.storage_units_per_word range  0 ..   15;
      command_status_AS     at  8*RP.storage_units_per_word range  8 ..   11;
      command_status_PS     at  8*RP.storage_units_per_word range 12 ..   15;
      command_status_addr   at  9*RP.storage_units_per_word range  0 ..   15;
      error_status_AS       at 10*RP.storage_units_per_word range  8 ..   11;
      error_status_PS       at 10*RP.storage_units_per_word range 12 ..   15;
      error_status_addr     at 11*RP.storage_units_per_word range  0 ..   15;
      input_index_AS        at 12*RP.storage_units_per_word range  8 ..   11;
      input_index_PS        at 12*RP.storage_units_per_word range 12 ..   15;
      input_index_addr      at 13*RP.storage_units_per_word range  0 ..   15;
      retries               at 14*RP.storage_units_per_word range  0 ..   15;
      new_poll_retries_AS   at 15*RP.storage_units_per_word range  8 ..   11;
      new_poll_retries_PS   at 15*RP.storage_units_per_word range 12 ..   15;
      new_poll_retries_addr at 16*RP.storage_units_per_word range  0 ..   15;
      terminal_ids_count    at 17*RP.storage_units_per_word range 11 ..   15;
      rt_ids_and_poll_seq   at 18*RP.storage_units_per_word range  0 ..2303;
   end record;
```

Figure 18. Representation Clause for Variant Record

```
                MODULE RESET_SYS_ENABLE_ROM
RESET_SYS_ENABLE_ROM CSECT ABSOLUTE=00070
INIT_STATUS     DATA 00000
                DATA 00000
                DATA 00000
TRANSFER_VECTOR CSECT ABSOLUTE=00040
                NOP                           ; SURMOS vectors 40
                BR    START_UP
                XIO   0,ESUR
BRANCH_SELF     BR    BRANCH_SELF             ; Vector to SURMOS at 44
START_UP        LISP  0,0F                    ; Fix up page registers
                LISP  1,0F
PR_LOOP         XIO   0,WIPR,1
                XIO   0,WOPR,1
                SISP  0,1
                SISP  1,1
                BGE   PR_LOOP
                LST   INIT_STATUS             ; Go to 0, start program
                END
```

**Figure 19.** Module Reset_Sys_Enable_ROM

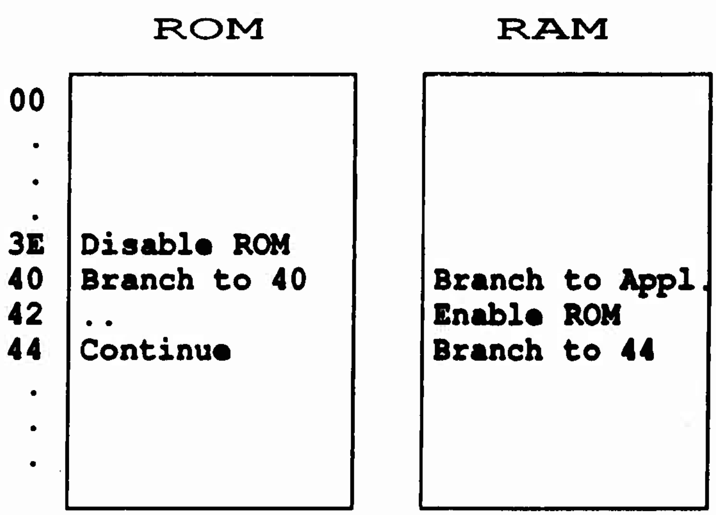


**Figure 20.** SURMOS Interface

have an instruction at location 42 to enable ROM (XIO 0, ESUR), followed by a branch to location 44 (BR BRANCH_SELF). Two alternate approaches were briefly investigated.

Since XIO instructions may be executed via Package Low_Level_IO, it would seem possible to code this module entirely in Ada, however, most of Reset_Sys_Enable_ROM executes before the Ada Run-Time System starts up. Low_Level_IO cannot be executed until after the page registers have been cleaned up and a stack has been established. The code that enables ROM, the only part of the module that does execute after the Run-Time System has started, must have the XIO instruction placed at location 42. This would not be possible, since the XIO would be executed from the body of a Low_Level_IO proce-

dure. Finally, the module would have had to start at an address smaller than 40 (hex), since there would have been some preliminary code before the first instruction in the procedure body. This would have overwritten the interrupt transfer vectors that must be in locations 20-3F (hex).

The other approach would have used machine-code insertion within an Ada subprogram. This would still be using assembly language, but would have had the advantage of "burying" it in an Ada module, which would put the module under the control of the Ada library system. Unfortunately, the compiler did not implement address clauses for subprograms, so linker instructions would have been required to start the module at location 40. This approach was rejected because it would still be programming in assembly language and would have required the additional complication of special linker instructions.

The remaining two lines of assembly language code were machine code insertions (see Figure 21). These instructions branch to location 40 (to restart the 11th Missile Application) and location 42 (to enable ROM, thereby terminating the application). Machine-code insertion was suitable because these instructions are executed after the Run-Time System has started and do not need to be in specific locations.

```
with Machine_Code;
use  Machine_Code;

package body System_Controller is

   procedure PreparetoDownload is
      begin   --  PreparetoDownload
         JC_Fmt'(Opcode => jc,
                 C      => unc,
                 RX     => R0,
                 Addr   => 16#0042#);
   end PreparetoDownload;

   procedure WarmStart is
      begin   --  WarmStart
         JC_Fmt'(Opcode => jc,
                 C      => unc,
                 RX     => R0,
                 Addr   => 16#0040#);
   end WarmStart;

end System_Controller;
```

**Figure 21.** Machine Code Insertion

## 3. USE OF OPTIONAL FEATURES

Optional ("Chapter 13") features of Ada were used extensively in the 11th Missile Application. The preceding discussion of the BIM Interface functions clearly shows that most of these features are not optional for real-time embedded applications. Table 26 lists the Chapter 13 features of Ada and indicates which ones were used. Comments on some of the options follow.

It was not necessary to specify the storage size of an array in order to force blank spaces between the components. Instead, the 11th Missile team did it by making the array components records and specifying the length of the records.

The 11th Missile team would have used the size representation clause for access types if it had been implemented. The BIM read 32-bit addresses, but the access types on the 1750A are 16 bits. This was gotten around by defining a 32-bit record containing the access type in the second 16 bits (see Figure 22).

Storage_size was used to specify the stack sizes of the tasks. This was crucial for the Navigation CSC and points up a useful programming hint: always specify tasks by defining a task type and then declaring an instance of the type, otherwise, storage size cannot be specified.

The 11th Missile team would have used the *small* attribute to fix the value of the least-significant-bit of fixed-point types if it had been implemented. Representations can be forced, however, by specifying the accuracy, range, and size of fixed-point types (see Figure 23).

If address specification for subprograms had been implemented, the 11th Missile Application may have been able to place the start-up code at location 40 (see Section V.2). Machine code insertion would still have had to been used, but at least the assembly language would have been buried in the body of an Ada subprogram.

It was not necessary to use PRAGMA Interface. In the cases where the code did interface to assembly language, it branched to the assembly statements with a machine-code insertion.

**TABLE 26. USE OF OPTIONAL ADA FEATURES BY 11TH MISSILE**

| 11th Missile Usage of Optional Ada Features | | |
|---|---|---|
| **Option** | **LRM** | **Used** |
| Length Clauses | 13.2 | |
| Size | | |
| Integer | | Y |
| Enumeration | | Y |
| Fixed Point | | Y |
| Floating Point | | N |
| Record | | Y |
| Array | | N |
| Access | | N |
| Storage_Size | | Y |
| Small | | N |
| Enumeration Representation | 13.3 | Y |
| Record Representation | 13.4 | |
| Alignment | | N |
| Component | | Y |
| Address | 13.5 | |
| Objects | | N |
| Subprogram | | N |
| Package | | N |
| Task | | N |
| Entry | | Y |
| Change of Representation | 13.6 | Y |
| Package System | 13.7 | |
| Named Numbers | 13.7.1 | Y |
| Representation Attributes | 13.7.2 | N |
| Representation Attributes of Real Types | 13.7.3 | N |
| Machine Code Insertion | 13.8 | Y |
| Interface to Other Languages | 13.9 | N |
| Unchecked Programming | 13.10 | |
| Unchecked_Deallocation | 13.10.1 | Y |
| Unchecked_Conversion | 13.10.2 | Y |

```
type long_input_message_ptrs is
   record
      dummy  : BIT.input_message_ptrs;
      ptr    : BIT.input_message_ptrs;
   end record;
```

Figure 22. Forcing a 32-bit Access Type

```
degrees_per_second_delta : constant := 2.0**(-6);
type degrees_per_second is delta degrees_per_second_delta
                        range -(2.0**9) .. (2.0**9)-degrees_per_second_delta;
for  degrees_per_second'size  use 16;
```

Figure 23. Forcing a Fixed-Point Representation without Using 'Small

# SECTION VI

# EVALUATION OF AN ADA COMPILER AND ITS USE IN THE 11TH MISSILE APPLICATION

The inability of the selected Ada/1750A compiler ("Compiler B") to correctly compile the CAMP parts and 11th Missile Application code caused program delays, reduced productivity, and ultimately, because of the lack of adequate work-arounds for some of the compiler errors, prevented hardware-in-the-loop testing of the Navigation CSC.

During the course of the 11th Missile Application development, many compiler errors were uncovered; additionally, a substantial number of deficiencies in the run-time performance of generated 1750A object code were uncovered. In some areas, the compiler functioned well, producing efficient code on a par with more mature Jovial compilers, but in other cases incorrect or inadequate code was produced.

The complicated semantics of the code very often proved to be too much for the compiler, which, although validated, was immature in the more advanced areas of the Ada language. For this reason, much effort was expended both generating Software Problem Reports for the compiler vendor, and devising work-arounds.

After compiler updates and modifications to the CAMP code produced executable software, the execution time and storage efficiency of the object code produced by the compiler were not acceptable for real-time embedded (RTE) applications. Unfortunately, the most powerful Ada features (tasking, generics, and variant records) were the least efficiently implemented. This implies that, for the time being, more sophisticated applications, including those using reusable software, will experience proportionally greater run-time performance degradation.

## 1. COMPILER PROBLEMS AND SOLUTIONS

The Ada compiler selected for the 11th Missile Application never compiled all of the Ada code in its original form. A workable load module could only be generated by "manually instantiating" numerous generics, manually in-lining some subprograms, and by combining separate compilation units.

### a. History of Compiler Utilization

"Compiler B" was initially selected for the 11th Missile Application. Benchmark tests of "Compiler B" showed the code generated by it to be as efficient as that produced by a JOVIAL compiler. However, it had two major problems. First, it was not validated and was not able to generate valid code for the very first 11th Missile unit test attempted. Second, it did not implement extended-precision floating-point, which was required for 11th Missile navigation and Kalman filtering functions.

For these reasons, the 11th Missile team switched to "Compiler A" in November, 1986. "Compiler A" was validated, but not mature enough to compile the CAMP parts. It also turned out to be extremely inefficient, generating object code modules 3 to 5 times larger than those produced by

"Compiler B". "Compiler A" (indeed, all Ada/1750A compilers at the time) limited object code to 64K-words instruction space plus 64K-words data space. It was not possible to fit the 11th Missile code into that amount of memory with such an inefficient compiler.

When it became apparent that there would be no quick solutions to "Compiler A's" problems (June, 1987), the 11th Missile team switched back to "Compiler B". By this time, "Compiler B" had been validated and extended-precision floating-point had been added. The compiler was still more efficient than "Compiler A". Another major consideration was that "Compiler B's" vendor provided much better (though much more expensive) maintenance support.

### b. Summary of Problem Reports

"Compiler B" was still not a mature product, however, as shown both by the number of problem reports submitted to the vendor and by the nature of the reports. Over the course of the 11th Missile development, 98 test cases were submitted to the vendor as 107 problem reports (some were submitted more than once).

Table 27 summarizes the problem reports by category. "Other" includes library errors, evaluation of compound boolean expressions, passing parameters to subprograms, code optimizer errors, etc. Although most of the reports involve advanced Ada features (e.g., tasking, generics), many of them were mundane (e.g., separate compilation of non-generic subunits, boolean expressions). The vendor's compiler fixes would sometimes reopen old problems. Sometimes, modifications would cause the compiler to fail the ACVC test suite. In short, "Compiler B" was not a mature, reliable product.

TABLE 27. PROBLEM REPORTS BY CATEGORY, "COMPILER B"

| Category | Number of Problems |
|---|---|
| Generics | 52 |
| Tasking | 12 |
| Visibility Rules | 10 |
| Separate Compilation (non-generic) | 6 |
| User Error | 4 |
| Excessive Storage | 3 |
| Other | 20 |

The frequency of problem reports did not decline with time. Figure 24 plots the number of problem reports submitted each month. The frequency of problem reports actually increased with time, but this is more a function of the amount of effort the 11th Missile team put into testing the software than the maturity of the compiler.

Figure 24 shows a pair of early problem reports submitted before the project switched to "Compiler A" (November, 1986). No further reports were submitted until the project began to consider switching back to "Compiler B". From April through August, 1987, the approach was to test a new compiler release, find the problems, submit test cases, and then wait for the next release. In the Fall of 1987, the strategy was switched to finding work-arounds to allow 11th Missile unit testing to proceed,

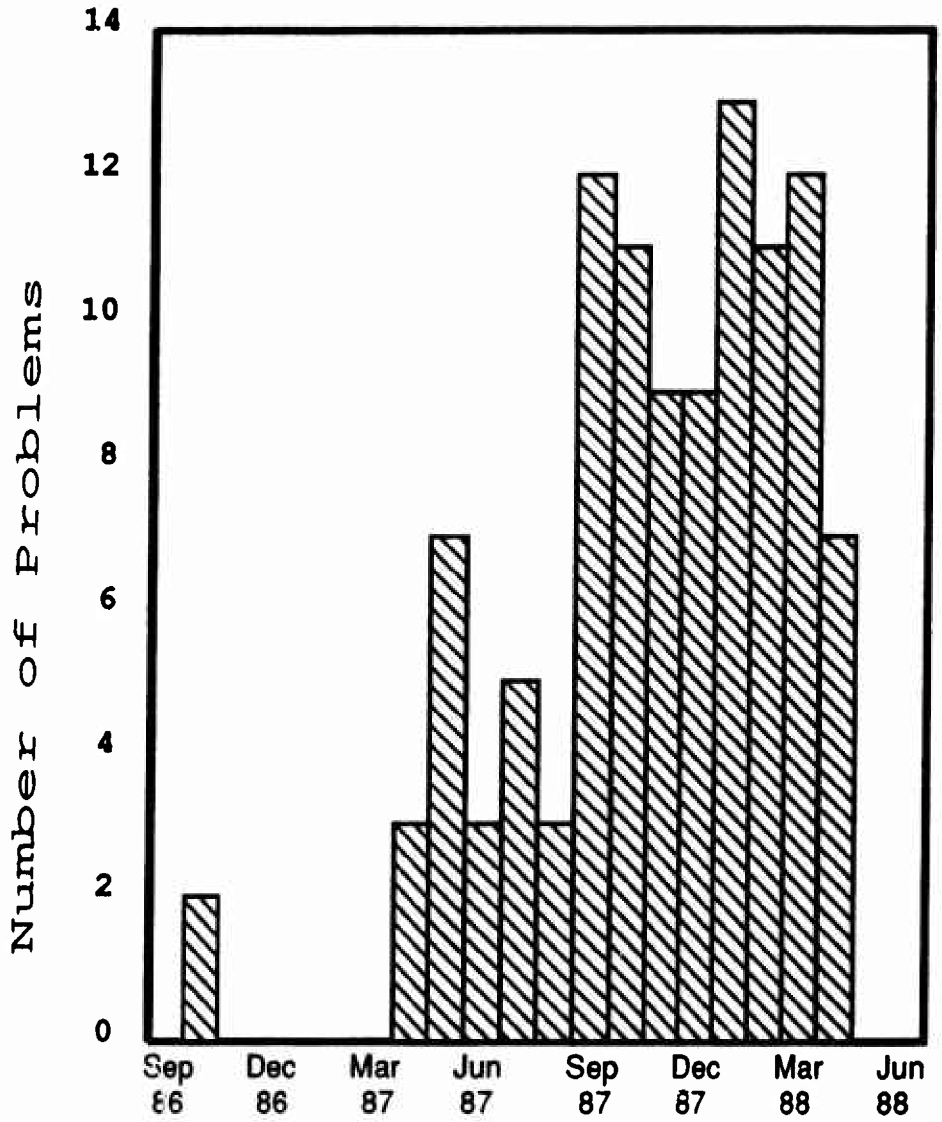


**Figure 24.** Problem Reports by Month, "Compiler B"

instead of waiting for the next compiler delivery. As a result, the problem submittal rate went up. But, within each of the two phases (Apr-Aug 87 and Sep 87-Apr 88), there is no clear trend; the number of problem reports neither rose nor declined.

Figure 25 shows the cumulative number of reports submitted, including those re-opened, and the number of problems fixed. The vertical distance between the lines is the number of open reports. The horizontal distance is the average time required to fix a problem. As can be seen, the average time to fix a problem increased with time.

MDAC-STL had an "on call" maintenance contract with the compiler vendor. This is a standard contract that allowed MDAC to consult by phone and to submit problem reports. The vendor sent one or more Ada system updates, as required, to MDAC between official releases. The vendor's response

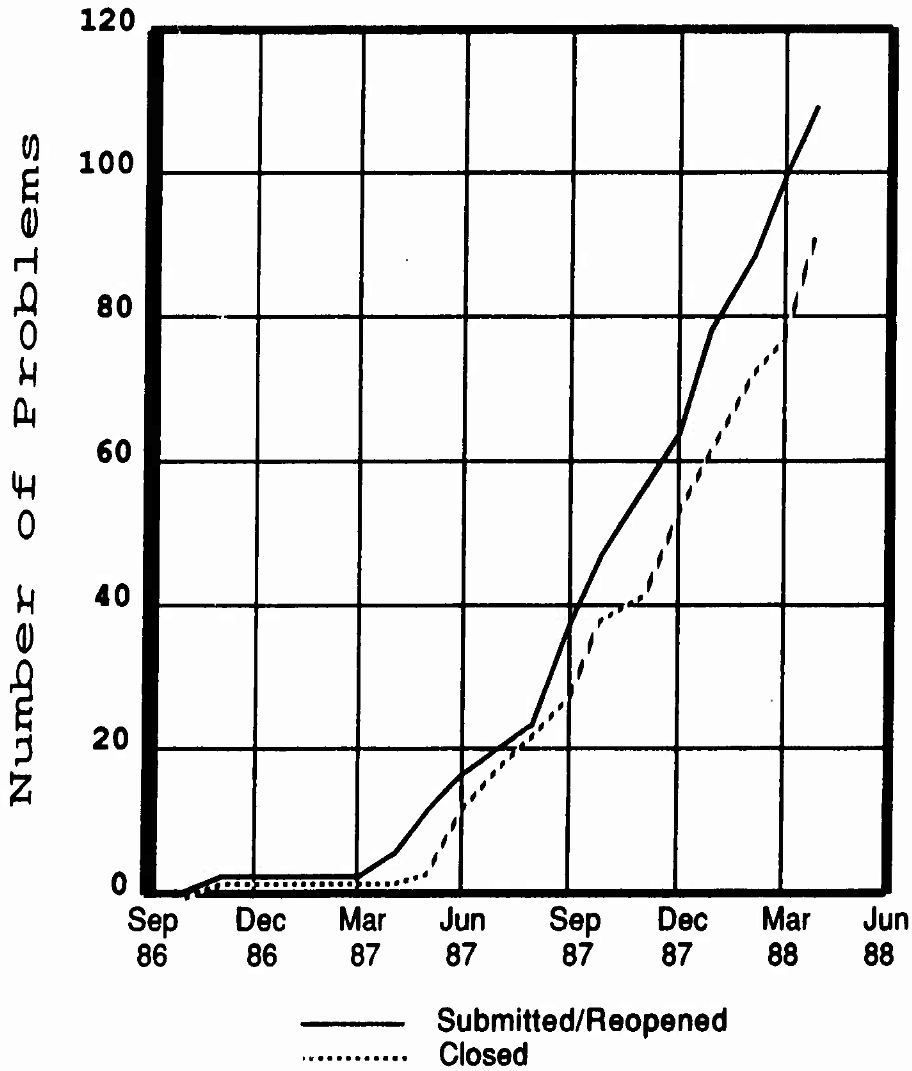


**Figure 25.** Cumulative History of Problem Reports, "Compiler B"

to problem reports was excellent. Nevertheless, over the life of the program, the average time to fix a problem was 37 calendar days (including week-ends and holidays). This shows that, until Ada/1750A compilers mature, an Ada project with critical schedule deadlines must put a compiler vendor under a special contract to fix problems quickly.

### c. Some Compiler Problems and Work-Arounds

Some of the problems encountered with "Compiler B" and work-arounds for them are explained in the following paragraphs.

#### (1) Generics

A major and recurring problem was "Compiler B's" inability to handle complex generics. Sometimes, all that was required to get the compiler to accept a generic was to combine separate compilation units. That is, the Ada body stubs within a generic unit's body were expanded in place by directly importing the required code. This simplified things for the compiler and often allowed the code to be compiled without errors, although there were some disadvantages to this. In addition to the time and effort required to reorganize the files, the physical organization of the software was disrupted, making up-to-date documentation more difficult to maintain and modifications more difficult to make. Furthermore, the simple expedient of combining separate subunits was often unsuccessful.

When combining subunits failed, a more extreme approach was to replace generic instantiations with customized "manual instantiations" of the associated code. This involved retrieving the software for a required generic and manually converting it to a non-generic. The code created was then placed into the software in place of the instantiation. Figures 26 and 27 give an example.

In many cases, manual instantiation was not a trivial process, since some logical implications of a generic instantiation are difficult or impossible to duplicate. For example, a generic instantiation may occur anywhere in a declarative part that a package, task, or subprogram specification may occur. Where such an instantiation occurs, a body for the instantiated unit is implicitly created and the elements of the unit are accessible according to the rules of Ada. However, when the unit is manually instantiated, a body is created, and that body cannot necessarily be inserted at the site of the former instantiation because of language rules. Since the manually created unit body of the former generic must usually be positioned at some distance from the manually created specification, the possibility arises that an element of the unit will be referenced prior to the necessary body elaborations. The chance that such error conditions would develop made the manual instantiation of generics a last resort.

Interestingly though, the insertion of customized code in place of generic instantiations usually brought about a savings in object code size and in operand memory utilization. (This is explained fur her in Section VI.2.) Indeed, even when the compiler successfully implemented generics, it was occasionally necessary to use manual instantiation to conserve storage space.

```
with Bus_Terminals,
     BIM_Interface_Types;
generic
   this_terminal              : in Bus_Terminals.terminals;
   bus_delay                  : in Standard.duration;
   with procedure Receive_Message
        (message : in BIM_Interface_Types.input_message_ptrs);
package RT_BIM_Interface is

   task type BIM_Interfaces is

-- -- ...

   end BIM_Interfaces;

   BIM_Interface : BIM_Interfaces;

end RT_BIM_Interface;


package body RT_BIM_Interface is

   task body BIM_Interfaces is

--  -- ...

   end BIM_Interfaces;

end RT_BIM_Interface;



with RT_BIM_Interface,
     Bus_Terminals;
package body Environment is

   package External_BIM is new RT_BIM_Interface
           (this_terminal   => Bus_Terminals.Nav;
            bus_delay       => 0.5;
            receive_message => Message_Manager.Receive_Message);

end Environment;
```

**Figure 26.** Code Before Manual Instantiation

## (2) Separate Subunits

Often, the compiler was unable to handle deeply nested separate compilations of unit bodies, even those which were not generic. This was generally fixed by combining separate subunits into one file as described above.

Figures 29 through 32 give a graphical representation of the effect of manual instantiations and separate body combination on the Guid_Computer software. Figure 28 explains the symbols used in the next four figures. The first two figures depict the baseline Guid_Computer procedure as well as the Guidance_Operations package it contains. The third and fourth figures show the same two units after being modified to manually instantiate generics and combine separate bodies.

```
-- Generic RT_BIM_Interface package no longer needed.

with Bus_Terminals,
     BIM_Interface_Types;
package body Environment is

   package External_BIM is

-- -- Object declarations and subprogram renamings used in place of the
-- -- generic actual parameters.

      this_terminal    : Bus_Terminals.terminals := Bus_Terminals.Nav;
      bus_delay        : Standard.Duration       := 0.5;
      procedure Receive_Message (message : in BIM_Interface_Types.input_message_ptrs)
              renames  Message_Manager.Receive_Message;

      task type BIM_Interfaces is     -- Code directly imported from the
                                      -- package specification of the
-- -- -- ...                         -- generic package RT_BIM_Interface.
                                      --
      end BIM_Interfaces;             --

      BIM_Interface : BIM_Interfaces;

   end External_BIM;


   package body External_BIM is     -- Body created to hold body code of the
                                    -- generic.

      task body BIM_Interfaces is   -- Code directly imported from the
                                    -- body of the generic package
-- -- -- ...                       -- RT_BIM_Interface.
                                    --
      end BIM_Interfaces;           --

   end External_BIM;

end Environment;
```

Figure 27. Code After Manual Instantiation

Table 28 shows the dramatic reduction in the use of generics due to manual instantiations. Table 29 shows that the number of files compiled also dropped dramatically as separate subunits were combined directly into their parent units.

### (3) Parameter Passing

Occasionally, even a simple parameter pass would not work. The solution to this problem, where feasible, was to manually in-line the code at the site of invocation. Figures 33 and 34 show an example of how this was done. Note that in the example, the names of the actual parameters used in the procedure call matched the names of the formal parameters. This eliminated the need to rename objects.

### (4) Machine Code Patches

Small problems in code generation, such as the use of an incorrect assembly language instruction, were sometimes solved by directly modifying the machine code in the load module. For example, a failure of the hardware-in-the-loop test of the Guid_Computer was directly attributable to the compiler's incorrect choice of shift instructions: an arithmetic shift had been used in place of a logical shift. This problem was corrected by altering the instruction word, a modification which permitted the test to run to completion without error.

### (5) Memory Utilization

Late in the project, the compiler was found to make extremely inefficient use of temporary operand storage space. Because of this, the 11th Missile Nav_Computer Software, which was larger than the Guid_Computer, could not initially be linked. When the static operand memory usage was reduced by software modifications, the successfully linked code still failed to run because of inefficiencies in the use of dynamic operand memory. This latter problem was caused, not by the explicit use of dynamic memory by the software, but by implicit uses of the heap by the generated object code. Also, the dynamic memory dedicated to the run-time stack was overused by inefficient allocations of temporary variable space. As a result, the working storage requirements of a subprogram or task were almost always more than had been anticipated during the design of the software.

One compiler-related space inefficiency, which was partially corrected by the vendor, was the failure to make thrifty use of temporary space. A subprogram, for example, required not only the operand space implied by the subprogram's local variables, but also required temporary space for each operator result. Because the 11th Missile incorporated large Kalman filter matrices, this was a serious deficiency which quickly caused the Nav_Computer to run out of operand space. The stack simply could not be made large enough to accommodate the inefficiency.

As a temporary solution to the problem, expressions containing operators with large results were sometimes placed within local procedures; Figure 35 shows an example of this. To a large extent, this limited the effect of the problem since the space set aside for the local procedures was immediately reclaimed after exiting those procedures.

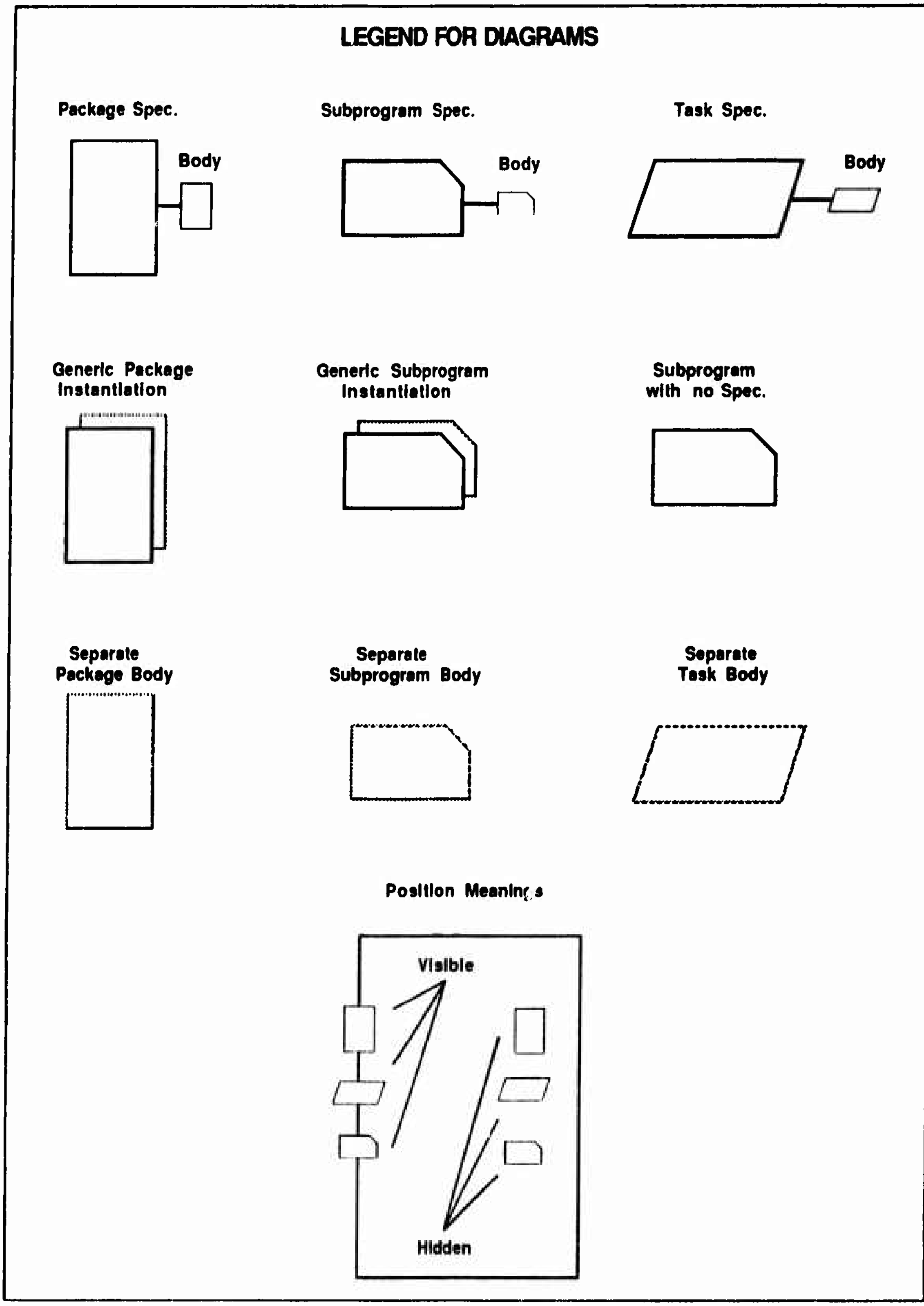


**Figure 28.** Legend for Diagrams

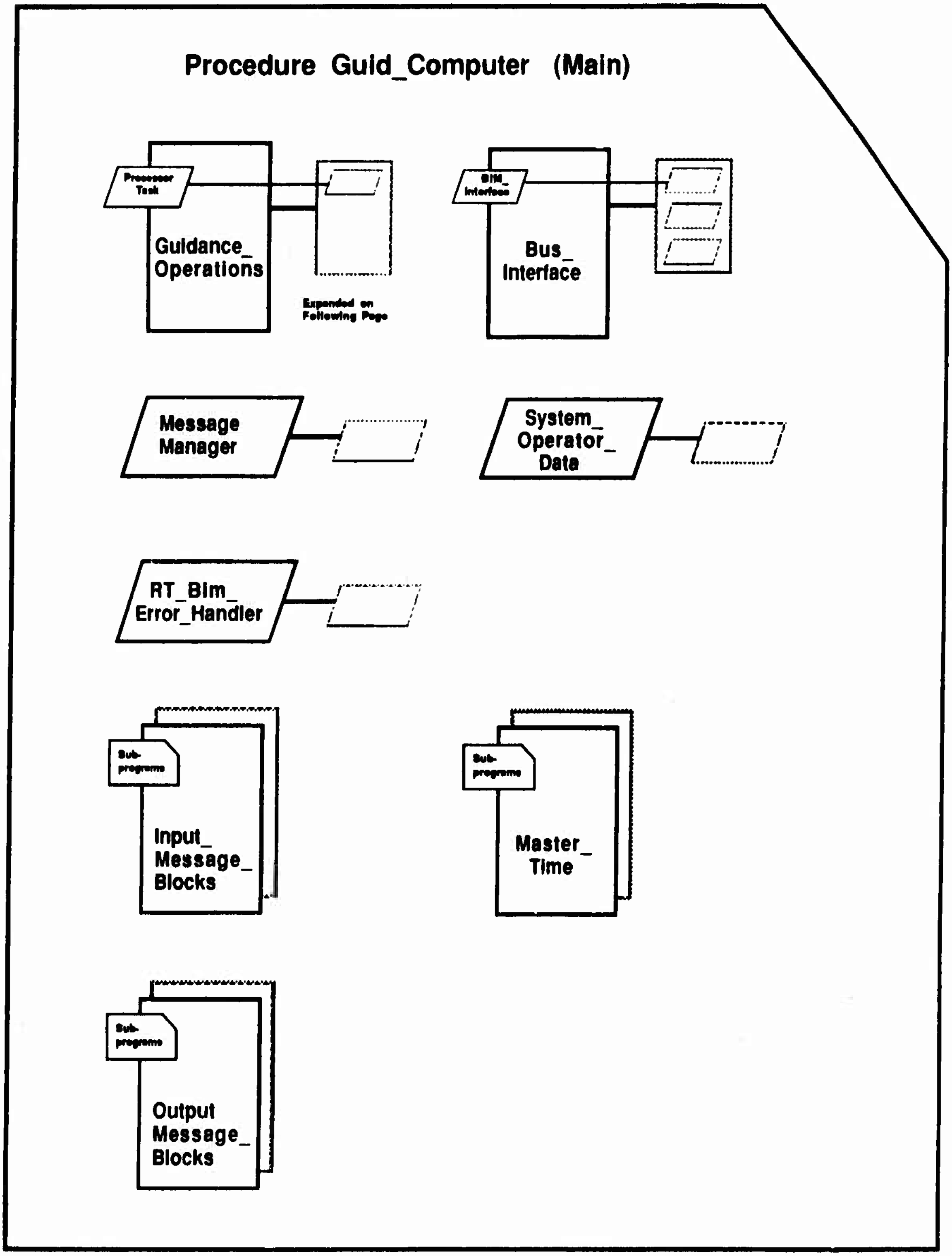


**Figure 29.** Baseline Guid_Computer Procedure

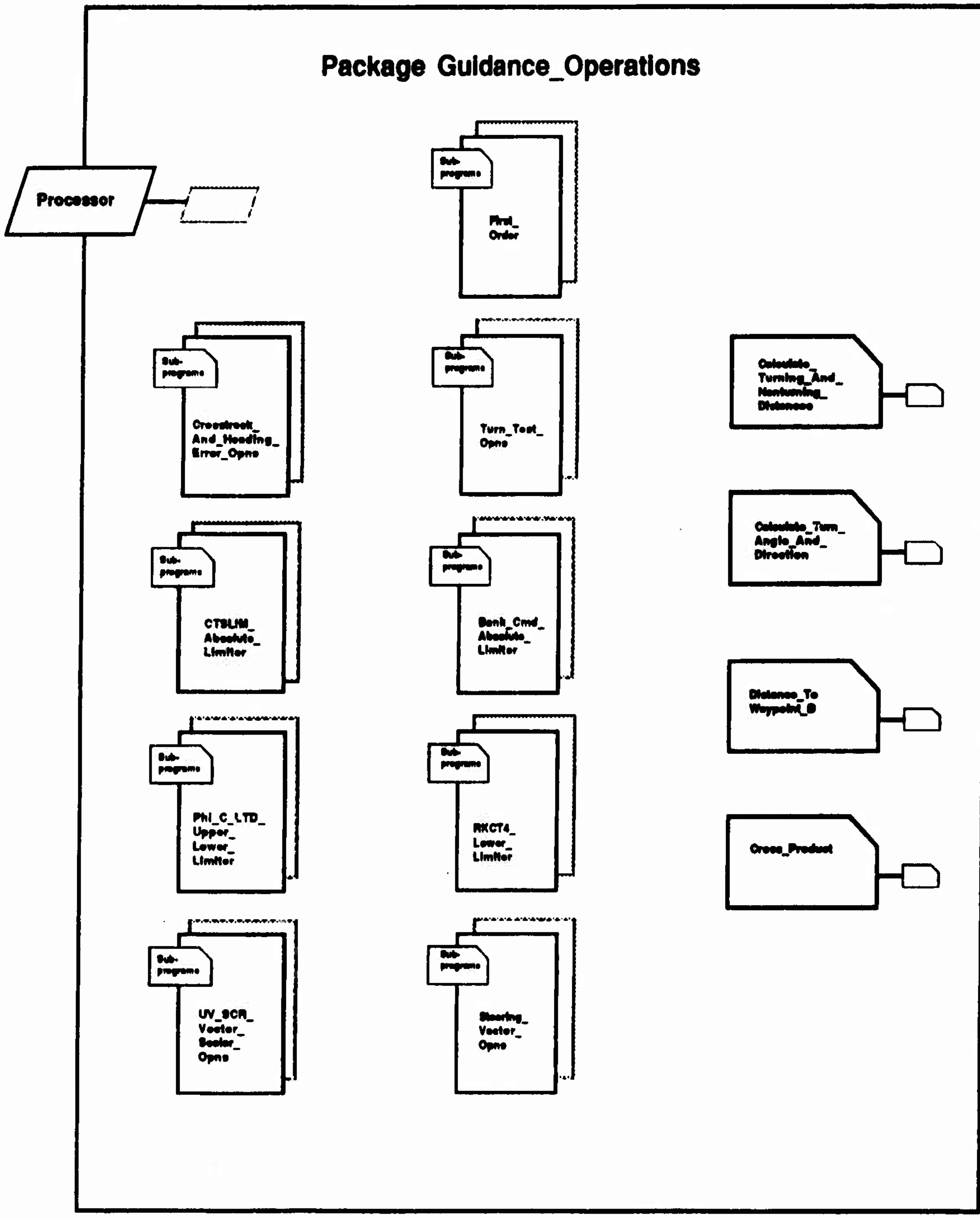


Figure 30. Baseline Guidance_Operations Package

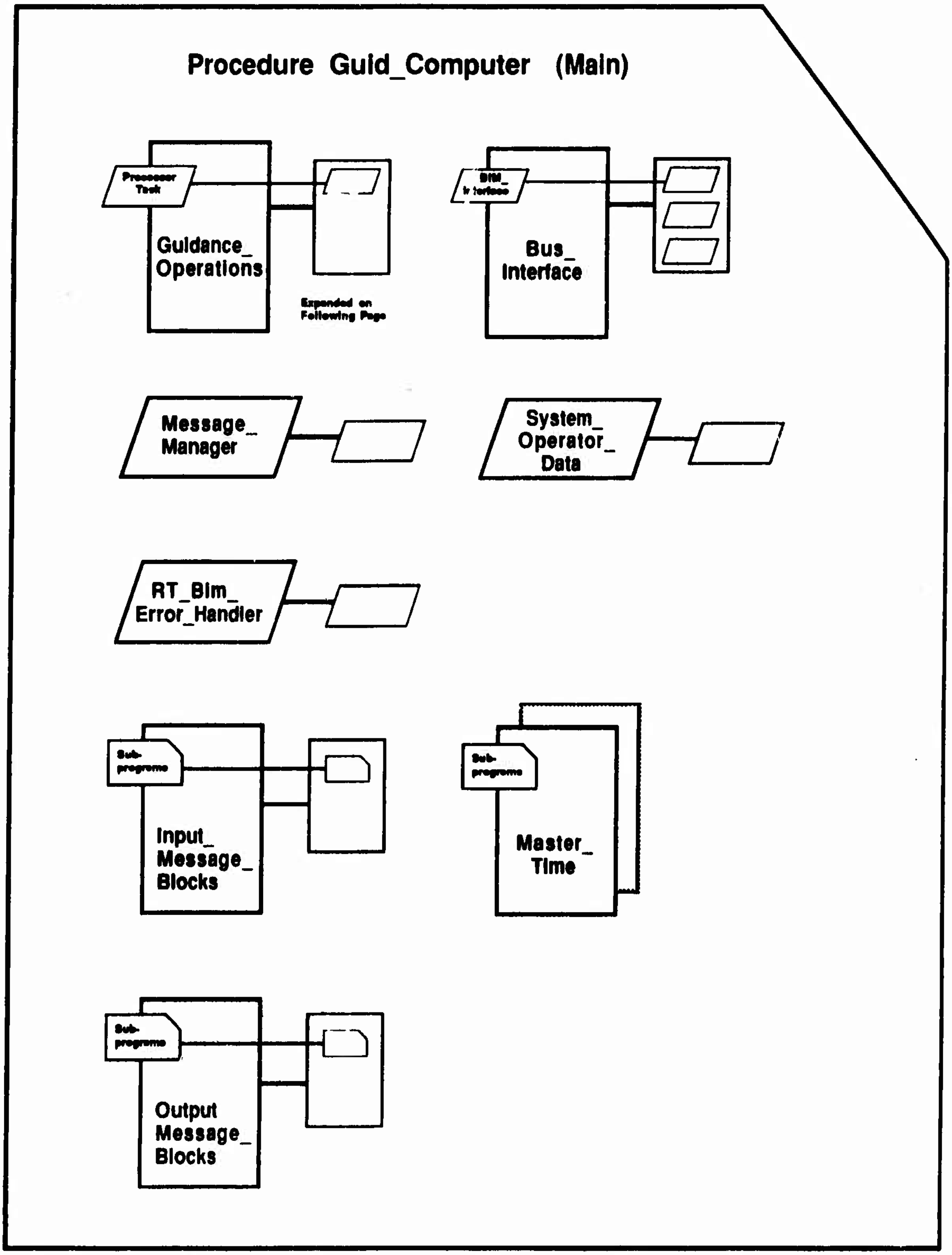


Figure 31. Modified Guid_Computer Procedure

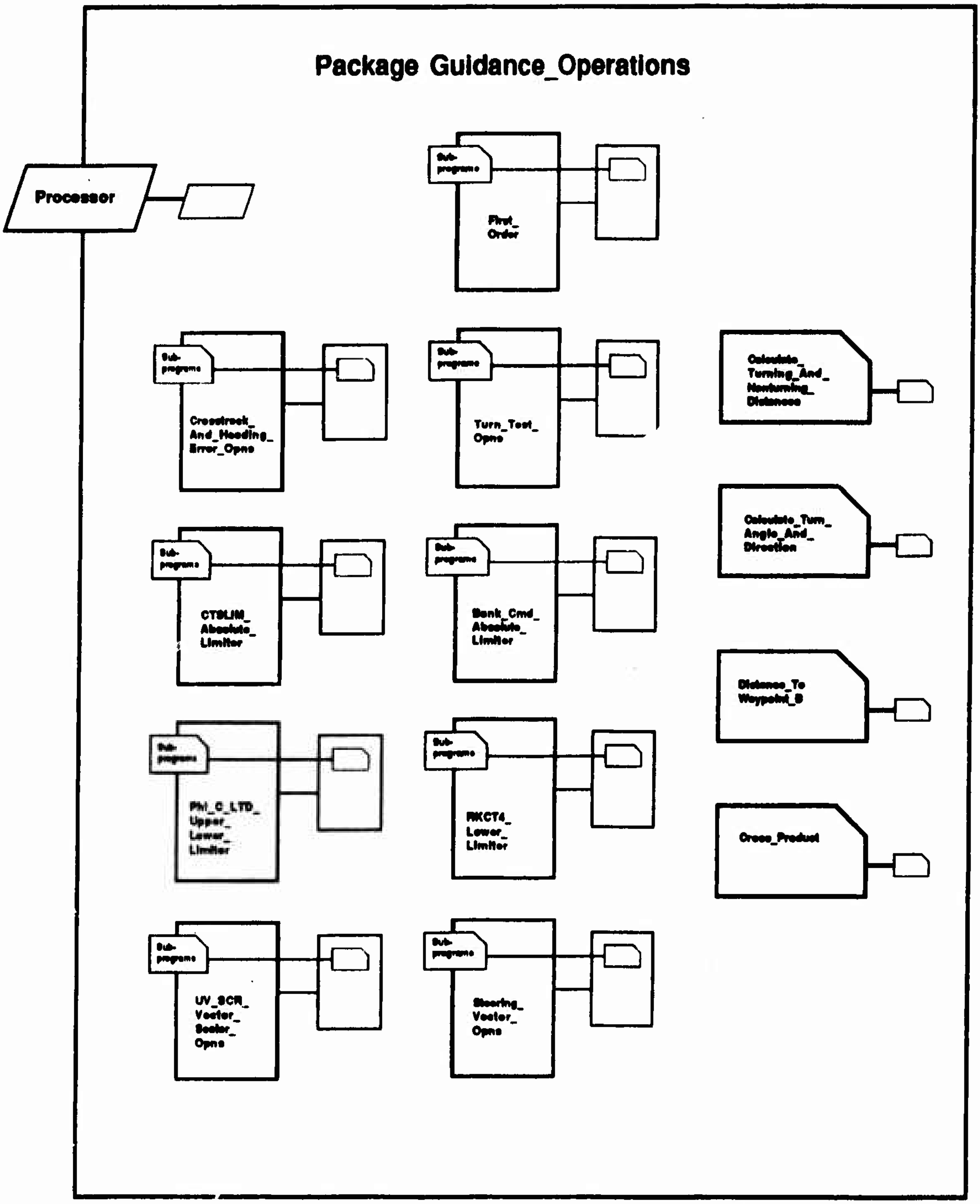


**Figure 32.** Modified Guidance_Operations Package

TABLE 28. USE OF GENERICS IN BASELINE AND MODIFIED SOFTWARE

| | Excluding Unchecked Conversion/Deallocation | | Including Unchecked Conversion/Deallocation | |
|---|---|---|---|---|
| | Baseline | Modified | Baseline | Modified |
| Guid_Computer Distinct Generics | 33 | 9 | 35 | 11 |
| Nav_Computer Distinct Generics | 73 | 49 | 75 | 52 |
| Raw Total Distinct Generics | 106 | 58 | 110 | 63 |
| Guid_Computer Instantiations | 49 | 16 | 74 | 43 |
| Nav_Computer Instantiations | 126 | 84 | 200 | 149 |
| Total Instantiations | 175 | 100 | 274 | 192 |

TABLE 29. SEPARATE SUBUNITS AND FILES COMPILED

| | Easeline | Modified |
|---|---|---|
| Guid_Computer Files | 89 | 59 |
| Nav_Computer Files | 213 | 177 |
| Total Files | 306 | 233 |
| Nav_Computer Separate Subunits | 128 | 98 |
| Guid_Computer Separate Subunits | 37 | 20 |
| Total Separate Subunits | 165 | 118 |

Later, the compiler vendor implemented a more efficient allocation scheme for temporary variables and the need for enclosing procedures was reduced. However, certain expressions, particularly very long or very complex expressions in the right-hand sides of assignment statements, still made excessive use of operand space. This problem proved to be far more tractable, and was remedied by breaking up long expressions into intermediate sub-expressions and using multiple assignment statements.

A second and more serious problem was the failure to reuse temporary operand space in the compiler's implementation of generics. While return values for ordinary operators and functions were placed on the stack and subsequently reclaimed, the return values of generic functions (and functions returning types obtained from generic instantiations) were placed in space dynamically allocated from heap. The heap space was never reclaimed and the heap was quickly exhausted. The primary work-around for this was the manual instantiation of offending objects, types, and operators. Nevertheless, the

```
   procedure Perform_Flight_Control_Operations
                 (Bank_Rate_Lim_Over_16_Sec : in      GDT.Radians_FP;
                  First_Pass                : in      BOOLEAN;
                  K                         : in      GDT.Single_Precision_Float;
                  Phi_C                     : in      GDT.Radians_FP;
                  Roll_Angle                : in      GDT.Radians_FP;
                  Phi_CLTD                  : in out GDT.Radians_FP;
                  Bank_Error                :    out GDT.Radians_FP;
                  Bank_Signal               :    out GDT.Volts) is

      Local_Bank_Error : GDT.Radians_FP;
      Phi_CLTDP        : GDT.Radians_FP;

   begin  --  Perform_Flight_Control_Operations

      if First_Pass then
         Phi_CLTDP := Roll_Angle;
      else
         Phi_CLTDP := Phi_CLTD;
      end if;

      Phi_CLTD := BankCmd_Absolute_Limiter.Limit (Signal => Phi_C);

      PhiCLTD_Upper_Lower_Limiter.Update_Limits
         (New_Upper_Limit => Phi_CLTDP + Bank_Rate_Lim_Over_16_Sec,
          New_Lower_Limit => Phi_CLTDP - Bank_Rate_Lim_Over_16_Sec);

-- -- -- ...

   end Perform_Flight_Control_Operations;


begin     -- Processor

--  Other statements...

   Perform_Flight_Control_Operations
                 (Bank_Rate_Lim_Over_16_Sec => Bank_Rate_Lim_Over_16_Sec
                  First_Pass                => First_Pass
                  K                         => K
                  Phi_C                     => Phi_C
                  Roll_Angle                => Roll_Angle
                  Phi_CLTD                  => Phi_CLTD
                  Bank_Error                => Bank_Error
                  Bank_Signal               => Bank_Signal              );

-- Other statements...

end Processor;
```

Figure 33. Section of Code Before Manual In-lining

damage caused by the problem was so extensive that several unit tests could not realistically be fixed. Completion of those tests was forced to await compiler modifications which the vendor made at the suggestion of the 11th Missile team.

```
begin     -- Processor

--  Other statements...

   Perform_Flight_Control_Operations : declare

      Local_Bank_Error : GDT.Radians_FP;
      Phi_CLTDP        : GDT.Radians_FP;

   begin  --  Perform_Flight_Control_Operations

   -- --limit the bank angle command to get Phi_CLTD

      if need_to_initialize then
         Phi_CLTDP := Roll_Angle;
      else
         Phi_CLTDP := Phi_CLTD;
      end if;

      Phi_CLTD := BankCmd_Absolute_Limiter.Limit (Signal => Phi_C);

   -- --update limiter with new minimum and maximum rate change
      PhiCLTD_Upper_Lower_Limiter.Update_Limits
         (New_Upper_Limit => Phi_CLTDP + Bank_Rate_Lim_Over_16_Sec,
          New_Lower_Limit => Phi_CLTDP - Bank_Rate_Lim_Over_16_Sec);

   -- -- ...

   end Perform_Flight_Control_Operations;

-- Other statements...

end Processor;
```

**Figure 34.** Section of Code After Manual In-lining

## 2. COMPILER INEFFICIENCY

The Ada/1750A cross-compiler, "Compiler B", used on 11th Missile was (as of May, 1988) in need of much improvement. Certain Ada constructs, especially the more powerful ones, were not implemented efficiently. Other constructs seemed to incorporate a good deal of optimization, yet the result was far from ideal. It is clear that the compiler developers are faced with a very challenging task.

### a. Tasking

The most costly throughput problem was the compiler's implementation of Ada tasking. In the Guid_Computer, 75% of processor throughput is expended in task rendezvous overhead while an additional 12% of throughput is expended for the remainder of the program. Although this processor-time consumption still allows the Guid_Computer to function within its real-time performance requirements, the high overhead of task rendezvous is serious. In the Nav_Computer, where tasking overhead is 138% of processor throughput, the real-time performance requirements are well out of reach.

```
procedure Example is      -- Space wasting.
   a, b, c, d, e, f, x : INTEGER;     -- Working storage size is 7 words.
begin

   x := a + b * c;      -- 2 more words allocated for operator results.

   x := d + e * f;      -- 2 more words allocated for operator results.

end Example;        -- Total working storage allocated = 11 words.


procedure Example is      -- Space saving.
   a, b, c, d, e, f, x : INTEGER;     -- Working storage size is 7 words.

   procedure space_saver1 is
   begin
      x := a + b * c;      -- 2 words allocated for operator results.
   end space_saver1;

   procedure space_saver2 is
   begin
      x := d + e * f;      -- 2 words allocated for operator results.
   end space_saver1;

begin

   space_saver1;

   space_saver2;

end Example;        -- Total working storage allocated = 9 words.
```

Figure 35. Saving Stack Space Using Enclosing Procedures

The compiler vendor blames this poor tasking run-time performance on Ada requirements for exception propagation. The language rules require that an exception occurring in the "do" clause of an Ada rendezvous be propagated to both tasks. The set-up for these exception contingencies is non-trivial and constitutes a large part of the excessive task rendezvous overhead.

Some of the throughput expense of tasking can be reclaimed by making careful selections of the kinds of tasking facilities used. For example, passing parameters to task entries was found to be expensive, as was the use of many-branched select statements and the use of guards before accept statements. Benchmark data indicates that overhead times grow quickly and proportionally with the number of accept statements used. In particular, guarded accept statements caused overhead times to grow; a greater time expense was incurred when the guards evaluated *true*. Parameter passing also caused a large increase in throughput consumption that was proportional to the number and kinds of parameters used. By minimizing parameter passing and the use of guards, a more reasonable execution rate could be obtained.

### b. Generics

While tasking proved to be the major inefficiency in throughput consumption, the use of generics was also very costly. As previously discussed, the use of heap space by generics wasted operand memory. Moreover, throughput was adversely affected by generic subprograms which carried a much greater run-time cost than their non-generic counterparts. Also, surprisingly, even instruction memory suffered from the use of generics.

"Compiler B" uses a single-body implementation of generics. One reentrant object code body is created and used for all instantiations of the generic. The idea was to save memory at the expense of throughput, which would be expected to rise for three reasons: use of the heap, worst-case assumptions for generic types, and translation between actual and formal types.

Generics had to use heap for any object whose size was not defined (e.g., an object whose generic formal type was private, or an unconstrained array, or an array whose index type was also a generic parameter). Although once the compiler had been fixed, space allocated for these objects was reclaimed when it is no longer necessary, the deallocation scheme inevitably fragmented the heap. Since real-time embedded systems can neither afford to perform extensive garbage collection nor allow wasted space, the use of generics was severely limited.

A small execution time cost was the assumption of "worst-case" actual types. For example, if a generic formal parameter was floating point, all operations on that type had to be implemented as extended floating point instructions, just in case the generic would be instantiated with an extended-float actual parameter.

With the "single-body" implementation, each instantiation of a generic unit results in the creation of a unique interface to the single body of object code. This interface translates from the actual type to the underlying implementation before invoking the body, and reverses the translation after the body completes. For example, the interface code might convert an input from float to extended-float. It was expected that this interface code would take additional execution time.

It was also expected (and stated in the literature of the compiler vendor) that the single-body implementation of generics would result in an over-all savings in instruction memory. This turned out not to be the case because the customized code created at each instantiation more than outweighed the space savings of the single-body method. For example, by manually instantiating the Kalman filter generics used by Nav_Computer, a total of 1877 words of instruction memory were saved. The savings obtained was not expected, since the Kalman_Filter made use of multiple instantiations of the same generics, exactly the circumstance under which the single-body method should have performed best. The major reason for this apparent paradox is the small size of the CAMP parts; as a result, the interface code was larger (sometimes several times larger) than the body of the generic.

The single-body method could be used in some cases to reduce memory usage. The compiler vendor needs to provide the user with the option to specify either multiple- or single-body generics. This could be done by implementing PRAGMA Optimize or PRAGMA Inline.

### c. Temporary Data Space

In the absence of a global optimizer, exception handling and constraint checking mandate the creation of a temporary object to store the value of the right-hand side of an assignment statement. (This temporary object may be a large data structure.) If an exception occurs while computing the right-hand side, all elements of the left-hand side must still be intact. Also, the results must be checked for conformance to type constraints prior to assignment to the object on the left-hand side. Either rule requires a temporary object.

Therefore, the use of large aggregates and functions returning large data structures wasted operand space. Language constructs of this kind were originally used because of the readability and naturalness of expression that they afford. For example, under the object-oriented design paradigm, a package body that contains a large matrix would permit this matrix to be read only via a function call that returns the matrix. Because of Ada language rules, the compiler creates intermediate temporary storage for the result of the function at invocation. The necessity for this temporary space makes this construct very inefficient, although it is also quite desirable.

Sufficiently comprehensive implementations of PRAGMAs Inline and Suppress would eliminate most of the space inefficiency in this case. Specifically, PRAGMA Suppress could be used to eliminate much of the need for temporary objects returned from function calls. PRAGMA Inline would, in fact, remove the function call entirely but, without PRAGMA Suppress, the Ada language rules would still require an intermediate temporary object.

Space problems excusable by Ada language requirements were often rectified by altering baseline code. These corrections were not considered work-arounds, therefore, but permanent changes to the software baseline.

### d. Other Causes of Inefficiency

The absence of PRAGMA Inline was often significant to the 11th Missile project. In many instances, the pragma could have been used to improve efficiency. For example, the readability of long algorithms was enhanced by factoring the Ada code into groups of subprograms. Also, because of object-oriented design, objects were hidden in package bodies and had to be accessed via function calls. Unfortunately, without PRAGMA Inline, a price was paid in terms of throughput, since overhead was introduced at the site of each subprogram call.

Variant records were costly since they required the compiler to generate code for storing them, comparing them, and altering discriminants. Unconstrained arrays contributed to heap management problems since space for them was allocated dynamically. The implementation of these two types of objects implied an unacceptably sophisticated use of memory for RTE software.

Finally, an important waste of memory space was the inclusion of unnecessary object code in load modules. The compiler used by the 11th Missile inserted all code of a required library unit in the program load module, even if only a part of the code was actually referenced. Such a waste of instruction space is intolerable for any real-time embedded application using the CAMP parts since many of them consist of large groups of generics bundled together into a package. These packages had to be modified and recompiled to eliminate unnecessary code, although a more sophisticated compiler/linker would eliminate the need to do this. For example, a compiler could separate the object code of subunits into separate object modules. A linker could then perform a module reference analysis to determine which modules could safely be excluded from a link.

The problems described here are particularly acute for real-time embedded code. Some would not be considered problems in general applications. For example, the VAX Ada compiler is generally regarded as excellent, even though it also includes unreferenced code. This is because generally a VAX is installed with megabytes of memory, compared to 128K words of memory for the 11th Missile Application.

### e. Are These Compiler Problems?

Some of the inefficiencies discussed above are arguably language problems rather than compiler problems. This is particularly true of task rendezvous times and the requirements for temporary storage of intermediate results.

There has been considerable discussion within the Ada community as to whether or not an efficient implementation of the general Ada task rendezvous is possible. It is certainly not possible without some sort of global optimizer. Similarly, the temporary space allocation problem can be solved only by a global optimizer that knows if there is an exception handler that will require the original value of the left hand side of the assignment. Discussions with "Compiler B's" vendor indicate that such an optimizer is years in the future.

Many of the problems appear to be a result of the complexity of Ada. At least in the short term, the complex rules imposed by Ada evidently leave the compiler implementor with quite a chore in simply

achieving conformance. Because of this, and because the Ada Compiler Validation Capability (ACVC) does not check for it, efficiency is often sacrificed to achieve validation. The complexity issue has often been raised as a criticism of Ada and, at least for the time being, it appears to be a significant factor in the industry's effort to develop efficient and effective RTE Ada compilers.

THIS REPORT HAS BEEN DELIMITED
AND CLEARED FOR PUBLIC RELEASE
UNDER DOD DIRECTIVE 5200.20 AND
NO RESTRICTIONS ARE IMPOSED UPON
ITS USE AND DISCLOSURE.

DISTRIBUTION STATEMENT A

APPROVED FOR PUBLIC RELEASE;
DISTRIBUTION UNLIMITED.

## SECTION VII

## CONCLUSIONS AND RECOMMENDATIONS

### 1. CONCLUSIONS

- The 11th Missile development demonstrated that a productivity improvement as high as 15% is possible for a navigation & guidance application using the CAMP parts, and that a productivity improvement as high as 28% is possible for this type of application using both the CAMP parts composition system Kalman Filter Constructor and the CAMP parts.

- The full productivity improvements from using a PCS, or even the parts, will not be realized until MIL-STD-1750A-targeted Ada compilers mature.

- The MIL-STD-1750A-targeted Ada compilers examined by the 11th Missile team are currently (May, 1988) inadequate for the following reasons:

  - *They do not handle complicated generics.* The CAMP parts utilize the most advanced generic features of Ada. The three validated Ada/1750A compilers tested by MDAC-STL either did not compile, or did not correctly execute, the CAMP parts.

  - *Generic instantiations are inefficient.* "Compiler B" used a single-body approach, which has significant execution-time and memory penalties. "Compiler A" used a multiple-body approach, even if the underlying base types are identical.

  - *PRAGMA Inline is not fully implemented.* "Compiler B" did not implement this pragma at all. Neither "Compiler B" nor "Compiler A" in-lined instantiations of generics. Since many of the parts are very small, there is a significant usage penalty if instantiations of generics cannot be put in-line.

  - *Task rendezvous are inefficient.* Over 75% of the Guid computer's and over 135% of the Navigation computer's throughput was required for task rendezvous.

- Ada is an effective language for real-time embedded software, provided that the optional ("Chapter 13") features of the language are implemented.

- The PCS Kalman Filter Constructor did not generate code suitable for use in the 11th Missile Application. The full-matrix code would have been too slow, while the sparse-matrix code exceeded the instruction memory limit.

- The PCS or the CAMP parts need to be extended into new areas, particularly:
    - Floating-point operators (the strong typing philosophy embodied by the CAMP parts means that many such operators are needed for an application)
    - Records and representation specifications for I/O ports or messages, and code to read from/write to those records
    - Code to sequence navigation functions
    - Code to sequence Kalman functions
- Parts in the following areas need to be modified or added:
    - Coordinate Vector Matrix Algebra
    - Wander Azimuth Navigation
    - Waypoint Steering
    - Kalman Filter
    - Signal Processing
    - Geometric Operations
    - WGS
    - Flat-earth Navigation

## 2. RECOMMENDATIONS

This Section recommends modifications to the CAMP parts, PCS, and Ada.

### a. Modifications to Parts

- Revisions to Coordinate_Vector_Matrix_Algebra (CVMA)
    - Change Matrix_Operations so that the two axes of type matrices can be different (see Figure 36). The most common use for the 3-by-3 matrices handled by CVMA is coordinate transformations. The old coordinate frame indexes the columns and the new frame indexes the rows.
    - Make similar modifications to Matrix_Scalar_Operations, Matrix_Vector_Multiply and Matrix_Matrix_Multiply (see Figures 37, 38, and 39).
    - Add parts to transpose a matrix and to multiply by the transpose of a matrix.
- Drop Compute_Latitude_Using_Arctangent, Compute_Longitude, and Compute_Wander_Azimuth_Angle from Wander_Azimuth_Navigation_Parts (WANav). These parts all use single-

Current

```
generic
   type Axes      is (<>);
   type Elements is digits <>;
package Matrix_Operations is

   type Matrices is array (Axes, Axes) of Elements;

                          .
                          .
                          .

end Matrix_Operations;
```

Recommended

```
generic
   type Row_Axes is (<>);
   type Col_Axes is (<>);
   type Elements is digits <>;
package Matrix_Operations is

   type Matrices is array (Row_Axes, Col_Axes) of Elements;

                          .
                          .
                          .

end Matrix_Operations;
```

**Figure 36.** Recommended Change to Matrix_Operations

parameter arctangents. (Versions using two-parameter arctangents are also in WANav.) The longitude and wander-azimuth parts give incorrect answers whenever the correct answer is in the second or third quadrant. In theory, the latitude part is correct, but all single-parameter arctangent functions lose accuracy near the poles. No military system can live with such restrictions; therefore, these parts should be dropped.

- Change Crosstrack_and_Heading_Error_Operations (part of Waypoint_Steering) to use two-parameter arctangent functions instead of single-parameter arctangents. The part is written with no assumptions on the range of the heading error, so, to be consistent, it should use full-range arctangent functions.
- Investigate every instance of a single-parameter arctangent in the parts. In each case, determine whether or not it should be changed to a two-parameter arctangent, or if an equivalent part that uses a two-parameter arctangent should be added.
- Changes to Kalman filter parts
  - Add a measurement reasonableness test that implements the following:

$$Reasonable := y^2 < M(hPh^T + r)$$

where Reasonable = true if measurement is acceptable

Current

```
generic
   type Axes       is (<>);
   type Elements1 is digits <>;
   type Elements2 is digits <>;
   type Scalars    is digits <>;
   type Matrices1 is array (Axes, Axes) of Elements1;
   type Matrices2 is array (Axes, Axes) of Elements2;
   with function "*" (Left  : Elements1;
                      Right : Scalars) return Elements2 is <>;
   with function "/" (Left  : Elements2;
                      Right : Scalars) return Elements1 is <>;
package Matrix_Scalar_Operations is

   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2;

   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1;

end Matrix_Scalar_Operations;
```

Recommended

```
generic
   type Row_Axes  is (<>);
   type Col_Axes  is (<>);
   type Elements1 is digits <>;
   type Elements2 is digits <>;
   type Scalars   is digits <>;
   type Matrices1 is array (Row_Axes, Col_Axes) of Elements1;
   type Matrices2 is array (Row_Axes, Col_Axes) of Elements2;
   with function "*" (Left  : Elements1;
                      Right : Scalars) return Elements2 is <>;
   with function "/" (Left  : Elements2;
                      Right : Scalars) return Elements1 is <>;
package Matrix_Scalar_Operations is

   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2;

   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1;

end Matrix_Scalar_Operations;
```

Figure 37. Recommended Change to Matrix_Scalar_Operations

y = $i^{th}$ element of the measurement vector, Y
M = tolerance
h = $i^{th}$ row of the measurement sensitivity matrix, H
P = system covariance matrix
r = $i^{th}$ diagonal element of the measurement covariance matrix, R

Two versions will be needed, one each for compact and complicated representations of H.

- Add Compute_Kalman_Gain parts that take advantage of intermediate products computed in the reasonableness test. Both the reasonableness test and Compute_Kalman_Gain compute $hPh^T + r$. In addition, Compute_Kalman_Gain computes $Ph^T$, which is an intermediate result

## Current

```
generic
   type Axes                   is (<>);
   type Input_Vector_Elements  is digits <>;
   type Output_Vector_Elements is digits <>;
   type Matrix_Elements        is digits <>;
   type Input_Vectors          is array (Axes) of Input_Vector_Elements;
   type Output_Vectors         is array (Axes) of Output_Vector_Elements;
   type Matrices               is array (Axes, Axes) of Matrix_Elements;
   with function "*" (Left  : Matrix_Elements;
                      Right : Input_Vector_Elements)
                     return Output_Vector_Elements is <>;
function Matrix_Vector_Multiply
            (Matrix : Matrices;
             Vector : Input_Vectors) return Output_Vectors;
```

## Recommended

```
generic
   type Axes1                  is (<>);
   type Axes2                  is (<>);
   type Input_Vector_Elements  is digits <>;
   type Output_Vector_Elements is digits <>;
   type Matrix_Elements        is digits <>;
   type Input_Vectors          is array (Axes2) of Input_Vector_Elements;
   type Output_Vectors         is array (Axes1) of Output_Vector_Elements;
   type Matrices               is array (Axes1, Axes2) of Matrix_Elements;
   with function "*" (Left  : Matrix_Elements;
                      Right : Input_Vector_Elements)
                     return Output_Vector_Elements is <>;
function Matrix_Vector_Multiply
            (Matrix : Matrices;
             Vector : Input_Vectors) return Output_Vectors;
```

Figure 38. Recommended Change to Matrix_Vector_Multiply

of the first computation. Versions of Compute_Kalman_Gain should be written that make use of these intermediate products from the reasonableness tests.

- Add simultaneous update parts. The current parts sequentially update the state and covariance, using one measurement element and one row of H at a time. A simultaneous part would update all elements/rows at once. A part to invert symmetric matrices will need to be added to General_Vector_Matrix_Algebra to support this.

- Revise the higher-level Kalman parts (Sequentially_Update_Covariance_Matrix_and_State_Vector and Kalman_Update) to take the procedures they will instantiate as arguments. This gives the user the option of specifying which procedure to use for Compute_Kalman_Gain, and Update_P.

• Signal_Processing Changes

  - Add a first-order filter that allows the user to specify the initial values of both the "previous input" and the "previous output" (these are stored in the filter software).

Current

```
generic
   type Axes            is (<>);
   type Left_Elements   is digits <>;
   type Right_Elements  is digits <>;
   type Result_Elements is digits <>;
   type Left_Matrices   is array (Axes, Axes) of Left_Elements;
   type Right_Matrices  is array (Axes, Axes) of Right_Elements;
   type Result_Matrices is array (Axes, Axes) of Result_Elements;
   with function "*" (Left  : Left_Elements;
                      Right : Right_Elements) return Result_Elements is <>;
function Matrix_Matrix_Multiply
            (Matrix1 : Left_Matrices;
             Matrix2 : Right_Matrices) return Result_Matrices;
```

Recommended

```
generic
   type Axes1           is (<>);
   type Axes2           is (<>);
   type Axes3           is (<>);
   type Left_Elements   is digits <>;
   type Right_Elements  is digits <>;
   type Result_Elements is digits <>;
   type Left_Matrices   is array (Axes1, Axes2) of Left_Elements;
   type Right_Matrices  is array (Axes2, Axes3) of Right_Elements;
   type Result_Matrices is array (Axes1, Axes3) of Result_Elements;
   with function "*" (Left  : Left_Elements;
                      Right : Right_Elements) return Result_Elements is <>;
function Matrix_Matrix_Multiply
            (Matrix1 : Left_Matrices;
             Matrix2 : Right_Matrices) return Result_Matrices;
```

Figure 39. Recommended Change to Matrix_Matrix_Multiply

- Change all filters to allow input and output to be different types.

- Add a third-order filter for barometric altitude smoothing.

• Add a part to Geometric_Operations that computes the geodetic coordinates (i.e., geodetic latitude and longitude) of a new point, given the geodetic coordinates of a starting point and the distance and heading from the starting point to the new point. This would be useful for mission-planning and ground analysis software.

• Add a set of WGS84 parts. WGS84 is supplanting WGS72 in some cruise missile applications.

• Add a set of "flat-earth" navigation parts. Very short range weapons (e.g., Have Slick) do not require navigation algorithms as sophisticated as those supplied by the parts.

### b. Modifications to the CAMP PCS

- Modify the Kalman Filter Constructor to add the following options:

  - An additional matrix representation. The constructor, as currently implemented, has two options for representing Kalman matrices: full or sparse. The full-matrix code use less instruction memory, but more operand memory and is relatively slow. The sparse-matrix code uses much more instruction memory, but is relatively fast. Neither was acceptable for the 11th Missile Application. Other methods for representing and manipulating sparse matrices should be investigated, and at least one additional method should be implemented in the PCS. Any method implemented should not be as code-intensive as the current sparse-matrix method, nor as data-intensive as the current full-matrix method.

  - Generate update control code for multi-measurement-type filters. The generated code would be equivalent to the Sequentially_Update_Covariance_Matrix_And_State_Vector part for single-measurement-type filters. The code would be similar to the body of the part, except that the measurement, measurement-variance, and measurement-sensitivity would be variant types (one variant for each type of measurement) and the body would contain a case statement with one branch for each variant.

  - Optionally include a measurement reasonableness test.

  - Optionally generate sequential or simultaneous update code.

- Enhance the Data Type Constructor to automatically generate floating-point operators as required. The CAMP parts are strongly typed, which means that many operators are needed. The Basic_Data_Types part has operators needed to instantiate the parts. The problem is that the new code written to use the parts needs many additional operators. At the very least, the constructor should write the body from the function specification. At best, it should also write the specification based on "missing function" compiler error messages.

- Add Input/Output constructors that:

  - Generate records and representation specifications for bus messages and I/O ports.

  - Generate functions to encode and decode bus messages.

  - Generates a skeleton select statement that invokes the message decoding functions.

- Modify the Navigation Constructors to generate an executive that invokes the instantiated parts.

### c. Suggested Ada Language Improvements

*Allow constant expressions in address clauses.* It was not possible to write a single generic package to handle both Bus-Controller and Remote-Terminal Bus Interface Modules (see section V.1.a), because the interrupt levels were different. The address representation clause, which is used to specify the interrupt level, must use a "simple_expression" for the address. In other words, it is impossible to make the interrupt address a generic parameter. Thus, the 11th Missile Application contains two very similar packages: one for Bus-Controller BIMs and one for Remote-Terminal BIMs.

*Allow representation specifications to be separated from the declarations they specify.* This would have permitted the 11th Missile team to create multiple sets of representation specifications, one set for each compiler used, instead of creating tools to comment out the specifications not applicable to the compiler in use (see Section II.2.f).

# THE APPENDIX

# 11TH MISSILE USAGE DATA BASE

## 1. INTRODUCTION AND BACKGROUND

One of the requirements of the 11th Missile Application was that it keep track of the CAMP parts used. Since a database which listed all the parts and their sizes (in lines of Ada code) already existed, a field to track 11th Missile usage was added to the tables that already stored the sizes. This field allowed a parts usage report to be generated from the original database relations. For a listing of the relations and their fields, see Volume I, Appendix A.

More than a simple list of parts used was needed: information about how a part was used, who used it, what project the user belonged to, and an additional remark also needed to be tracked. This additional information was stored in a separate relation, called the Parts_Usage relation.

The Usage table and its fields are shown in Table A-1.

TABLE A-1. PARTS USAGE FIELDS AND DESCRIPTIONS

| Parts Usage Relation | |
|---|---|
| **Column Name** | **Description** |
| User_Part | The part number of the application or CAMP part which makes use of the part. |
| Part_Used | The part number of the CAMP part being used. |
| Project | The project name of the user_part. |
| Usage_Type | The type of usage. Types included direct, modified, and indirect. |
| Remark | A special remark characterizing or giving additional comments about the parts usage. |

It should be noted that the User_Part and Part_Used fields relate to the part numbers that are in the *TLCSC* and *Adalevel* relations. This allows applications access to the information stored in these tables. At present, since only sizes are required in the reports generated, and the sizes can be gotten more easily from the *TLCSC* and *Adalevel* relations, there are no specialized reports making use of the usage table.

## 2. DATABASE ISSUES

There were several issues concerning the Usage database which were distinct from the size database. The first was how to count modified code. The 11th Missile Application counts a part as used if it modifies the part for its own usage. Modification, of course, changes the line count of the part. The database, however, maintains the line count of the original part, not the modified count. In counting code for the 11th Missile Application parts usage, it was preferable to count the modified lines of code for modified parts. This meant that these numbers had to be put into the reports manually. It didn't seem feasible to put an extra field in the database, since the number of parts actually modified was small.

Another issue was the type of usage. The parts are heavily inter-dependent (i.e., the parts use other

parts). If the 11th Missile Application used a part which in turn used another part, then both of those parts needed to be marked as used. This means that the parts usage of parts must be derived from the 11th Missile usage list. This can be done automatically from the *Parts_Usage* relation, but not from the *TLCSC* and *Adalevel* relations. Since the usage information is duplicated in all these tables, it is possible to have inconsistent data.

A third difficulty arose from the nature of the parts. Since a part could be a TLCSC, an LLCSC, or a unit, how to count lines of code became a problem. When the part is a unit, often the specification for the part is in a separate file, since the CAMP parts make extensive use of separate compilation of Ada units. The question then arises as to whether the specification should be counted or not. If it is counted, it may contain the specifications of other pieces of a part which were not actually used, along with the specification that was actually used. If it is not counted, then the specification code for the piece actually used doesn't get counted. This was handled on a case-by-case basis . If all, or most, of the parts in a package were used, the specification code count was included. If only a few parts from a package were used, the specification code count was not included.

## 3. PARTS USAGE AND CODE COUNT

The parts usage and code count table (see Table A-2) is an edited version of the parts usage table from Volume I, Appendix A. The comment size, test code size, and 11th Missile usage columns were deleted, and the "Use Code" column added. Every part has one of the following five usage codes assigned:

- **U** (*Used*): The part was used without modification.
- **M** (*Modified*): The part was used with modification.
- **DF** (*Duplicate Function*): The part implements the same function as a part that is used by the 11th Missile.
- **NA** (*Not Applicable*): The part implements a function not required by the 11th Missile.
- **NC** (*Not Compatible*): The part performs a function required by the 11th Missile, but was not used.

All part code sizes were deleted from the table, except for those parts used without modification. The code size totals were computed manually.

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 1 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| F001 | Common Navigation Parts | | | N | |
| | Altitude Integration | 12 | 7 | Y | U |
| | Reinitialize | 2 | 7 | N | |
| | Integrate | 3 | 13 | N | |
| | Compute Ground Velocity | 10 | 8 | Y | U |
| | Compute Gravitational Acceleration Lat In | | | Y | DF |
| | Compute Gravitational Acceleration Sin Lat In | 19 | 13 | Y | U |
| | Compute Heading | | | Y | NA |
| | Update Velocity | 20 | 8 | Y | U |
| | Reinitialize | 1 | 5 | N | |
| | Update | 4 | 16 | N | |
| | Current Velocity | 1 | 5 | N | |
| | Scalar Velocity | | | Y | NA |
| | Compute Rotation Increments | | | Y | NA |
| | SUBTOTALS | 72 | 82 | 8 | |
| F002 | Wander Azimuth Navigation Parts | | | N | |
| | Compute Earth Relative Horizontal Velocities | | | Y | DF |
| | Compute Total Angular Velocity | | | Y | NA |
| | Compute Coriolis Acceleration | 19 | 12 | Y | U |
| | Total Platform Rotation Rate | 11 | 9 | Y | U |
| | Earth Rotation Rate | 12 | 7 | Y | U |
| | Compute | 4 | 11 | N | |
| | Compute Earth Relative Navigation Rotation Rate | 18 | 15 | Y | U |
| | Compute Wander Azimuth Angle | | | Y | DF |
| | Compute Latitude | | | Y | DF |
| | Compute Latitude Using Arctan | | | Y | DF |
| | Compute East Velocity with Sin Cos In | 11 | 13 | Y | U |
| | Compute Longitude | | | Y | DF |
| | Compute Curvatures | 31 | 30 | Y | U |
| | Compute East Velocity | | | Y | DF |
| | Compute North Velocity | | | Y | DF |
| | Coriolis Acceleration from Total Rates | | | Y | NA |
| | Compute | | | N | |
| | Compute North Velocity with Sin Cos In | 11 | 13 | Y | U |
| | Compute Earth Relative Horizontal Velocities with Sin Co | | | Y | NA |
| | Compute Latitude Using Two Value Arctangent | 14 | 16 | Y | U |
| | Compute Longitude using Two Value Arctangent | 11 | 11 | Y | U |
| | Compute Wander Azimuth Angle using Two Value Arctangent | 10 | 12 | Y | U |
| | SUBTOTALS | 152 | 149 | 20 | |
| F003 | North Pointing Navigation Parts | | | N | |
| | Compute Coriolis Acceleration | | | Y | DF |
| | Total Platform Rotation Rates | | | Y | DF |
| | Earth Rotation Rate | | | Y | DF |
| | Compute | | | N | |
| | Earth Relative Navigation Rotation Rate | | | Y | DF |
| | Compute | | | N | |
| | Latitude Integration | | | Y | DF |
| | Reinitialize | | | N | |
| | Integrate | | | N | |
| | Longitude Integration | | | Y | DF |
| | Reinitialize | | | N | |
| | Integrate | | | N | |
| | Radius of Curvature | | | Y | DF |
| | Compute | | | N | |
| | SUBTOTALS | 0 | 0 | 7 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 2 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P361 | General Utilities | | | N | |
| | Instruction Set Test | | | Y | NA |
| | SUBTOTALS | 0 | 0 | 1 | |
| P601 | Asynchronous Control | | | Y | NA |
| | Data Driven Task Shell | | | N | |
| | Interrupt-Driven Task Shell | | | N | |
| | Aperiodic Task Shell | | | N | |
| | Continuous Task Shell | | | N | |
| | Periodic Task Shell | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |
| P602 | Communication Parts | | | [illegible] | |
| | Update Exclusion | | | [illegible] | |
| | Read Update | | | [illegible] | |
| | Attempt Read | | | N | |
| | Attempt Read Wait | | | N | |
| | Attempt Read Delay | | | N | |
| | Attempt Start Update | | | N | |
| | Attempt Start Update Wait | | | N | |
| | Attempt Start Update Delay | | | N | |
| | Attempt Complete Update | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |
| P611 | WGS72 Ellipsoid Metric Data | 0 | 0 | Y | M |
| P612 | WGS72 Ellipsoid Engineering Data | 0 | 0 | Y | M |
| P613 | WGS72 Ellipsoid Unitless Data | 11 | 0 | Y | U |
| P614 | Conversion Factors | 41 | 0 | Y | U |
| P615 | Universal Constants | 9 | 0 | Y | U |
| P621 | Basic Data Types | 0 | 0 | Y | M |
| P622 | Kalman Filter Data Types | 0 | 0 | Y | MC |
| P623 | Autopilot Data Types | 0 | 0 | Y | NA |
| P631 | Missile Radar Altimeter Handler Parts | | | Y | NA |
| | Power On | | | N | |
| | Power Off | | | N | |
| | Goto Transmit Mode | | | N | |
| | Goto Standby Mode | | | N | |
| | Perform Built In Test | | | N | |
| | Perform Built In Test Sequence | | | N | |
| | Read Altitude Feet | | | N | |
| | Read Altitude Integer | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 3 of 18)

| TLCSC Number | TLCSC Name<br>Lower Level Units | Code Size<br>spec | <br>body | Part | Use Code |
|---|---|---|---|---|---|
| P632 | Missile Radar Altimeter Handler Auto | | | Y | NA |
| | Goto Transmit Mode | | | N | |
| | Goto Standby Mode | | | N | |
| | Perform Built In Test | | | N | |
| | Perform Built In Test Sequence | | | N | |
| | Read Altitude Feet | | | N | |
| | Read Altitude Integer | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |
| P633 | Bus Interface Parts | | | Y | M |
| | Send Message Using Address No Wait | | | N | |
| | Send Message Using Address Wait | | | N | |
| | Data Transfer No Wait | | | N | |
| | Data Transfer Wait | | | N | |
| | Perform Built In Test | | | N | |
| | Interface | | | N | |
| | Update Retry Count | | | N | |
| | Send Command Wait | | | N | |
| | Send Message No Wait | | | N | |
| | Send Message Wait | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |
| P634 | Clock Handler | 5 | 11 | Y | U |
| | Current Time | 1 | 5 | N | |
| | Converted Time | 2 | 6 | N | |
| | Reset Clock | 1 | 5 | N | |
| | Synchronize Clock | 3 | 7 | N | |
| | Elapsed Time | 1 | 9 | N | |
| | SUBTOTALS | 13 | 43 | 1 | |
| P644 | Direction Cosine Matrix Operations | 5 | 5 | N | |
| | DCM General Operations | 11 | 11 | N | |
| | DCM Initialized From Reference | 23 | 62 | Y | U |
| | DCM Trapezoidal Integration | | | Y | DF |
| | Reinitialize Angular Velocities | | | N | |
| | Perform_Trapezoidal_Integration of DCM | | | N | |
| | Perform Rectangular Integration of DCM | 24 | 27 | Y | U |
| | Reorthonormalize DCM | 23 | 33 | Y | U |
| | Frame Misalignment | 29 | 18 | Y | U |
| | Aligned DCM Matrix | 29 | 26 | Y | U |
| | DCM From Quaternion | 26 | 37 | Y | U |
| | Compute First Row from Orthonormal | 16 | 11 | Y | U |
| | CNE Operations | 29 | 26 | N | |
| | Reorthonormalize CNE | 1 | 4 | Y | U |
| | CNE Initialized From Earth Position | 15 | 28 | Y | U |
| | CNE Integration | 14 | 21 | Y | U |
| | Perform Trapezoidal Integration of CNE | 5 | 9 | N | |
| | Reinit Ang Vel For Trapez Integ of CNE | 3 | 8 | N | |
| | Perform Rectangular Integration of CNE | 5 | 7 | N | |
| | Alignment Parts | 18 | 21 | Y | U |
| | Frame Misalignment of CNE | 4 | 7 | N | |
| | Aligned CNE Matix | 4 | 7 | N | |
| | CNE From Quaternion | 13 | 11 | Y | U |
| | Compute CNE | 3 | 6 | N | |
| | Compute First Row of CNE From Orthonormal | 2 | 5 | Y | U |
| | CNE Initialized From Reference | 8 | 17 | Y | U |
| | SUBTOTALS | 310 | 407 | 15 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 4 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P651 | Kalman Filter Common Parts | | | N | |
| | State Transition And Process Noise Matrices Manager | 31 | 10 | Y | U |
| | Initialize | 1 | 6 | N | |
| | Propagate | 3 | 10 | N | |
| | Get_Current | 3 | 7 | N | |
| | Propagated_Phi | 1 | 5 | N | |
| | Error Covariance Matrix Manager | 15 | 8 | Y | U |
| | Initialize | 1 | 5 | N | |
| | Propagate | 2 | 6 | N | |
| | P | 1 | 5 | N | |
| | State Transition Matrix Manager | | | Y | DF |
| | Propagated_Phi | | | N | |
| | Initialize | | | N | |
| | Propagate | | | N | |
| | SUBTOTALS | 58 | 62 | 3 | |
| P652 | Kalman Filter Compact H Parts | | | N | |
| | Compute Kalman Gain | 19 | 18 | Y | U |
| | Update Error Covariance Matrix | | | Y | DF |
| | Update State Vector | 20 | 13 | Y | U |
| | Sequentially Update Covariance Matrix and State Vector | | | Y | NC |
| | Update | | | N | |
| | Kalman Update | | | Y | NC |
| | Update | | | N | |
| | Update Error Covariance Matrix General Form | 29 | 17 | Y | U |
| | SUBTOTALS | 68 | 48 | 6 | |
| P653 | Kalman Filter Complicated H Parts | | | N | |
| | Compute Kalman Gain | 30 | 20 | Y | U |
| | Update Error Covariance Matrix | | | Y | DF |
| | Update State Vector | 26 | 14 | Y | U |
| | Sequentially Update Covariance Matrix and State Vector | | | Y | NC |
| | Update | | | N | |
| | Kalman Update | | | Y | NC |
| | Update | | | N | |
| | Update Error Covariance Matrix General Form | 35 | 17 | Y | U |
| | SUBTOTALS | 91 | 51 | 6 | |
| P661 | Waypoint Steering | | | N | |
| | Distance to Current Waypoint | | | Y | DF |
| | Compute Turning and Nonturning Distances | 12 | 14 | Y | U |
| | Turn Test Operations | 5 | 14 | Y | U |
| | Stop Test | 4 | 13 | N | |
| | Start Test | 4 | 13 | N | |
| | Steering Vector Operations | | | Y | DF |
| | Initialize | | | N | |
| | Update | | | N | |
| | Steering Vector Operations with Arcsin | 24 | 23 | Y | U |
| | Initialize | 12 | 40 | N | |
| | Update | 8 | 23 | N | |
| | Compute Turn Angle and Direction | 18 | 24 | Y | U |
| | Crosstrack and Heading Error Operations | | 33 | Y | M |
| | Compute When not Turning | | | N | |
| | Compute | | | N | |
| | Compute When Turning | 11 | 43 | N | |
| | Distance to Current Waypoint with Arcsin | 19 | 11 | Y | U |
| | SUBTOTALS | 117 | 251 | 8 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 5 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P662 | Autopilot | | | N | |
| | Integral Plus Proportional Gain | | | Y | NA |
| | Integrate | | | N | |
| | Update Proportional Gain | | | N | |
| | Pitch Autopilot | | | Y | NA |
| | Initialize Pitch Autopilot | | | N | |
| | Compute Elevator Command | | | N | |
| | Update Pitch Rate Gain | | | N | |
| | Update Acceleration Gain | | | N | |
| | Update Integrator Gain | | | N | |
| | Update Integrator Limit | | | N | |
| | Update Proportional Gain | | | N | |
| | Lateral Directional Autopilot | | | Y | NC |
| | Initialize Lateral Directional Autopilot | | | N | |
| | Compute Aileron Rudder Commands | | | N | |
| | Update Aileron Integrator Gain | | | N | |
| | Update Aileron Integrator Limit | | | N | |
| | Update Roll Command Proportional Gain | | | N | |
| | Update Roll Rate Gain For Aileron | | | N | |
| | Update Yaw Rate Gain For Aileron | | | N | |
| | Update Rudder Integrator Gain | | | N | |
| | Update Rudder Integrator Limit | | | N | |
| | Update Feedback Rate Gain For Rudder | | | N | |
| | Update Roll Rate Gain For Rudder | | | N | |
| | Update Acceleration Proportional Gain | | | N | |
| | SUBTOTALS | 0 | 0 | 3 | |
| P671 | Air Data Parts | | | N | |
| | Compute Outside Air Temperature | | | Y | NA |
| | Compute Pressure Ratio | | | Y | NA |
| | Compute Mach | | | Y | NA |
| | Compute Dynamic Pressure | | | Y | NA |
| | Compute Speed of Sound | | | Y | NA |
| | Barometric Altitude Integration | | | Y | NA |
| | Compute Barometric Altitude | | | N | |
| | SUBTOTALS | 0 | 0 | 6 | |
| P672 | Fuel Control Parts | | | N | |
| | Throttle Command Manager | | | Y | NA |
| | Compute Throttle Command | | | N | |
| | Update Mach Error Limit | | | N | |
| | Update Mach Error Integral Limit | | | N | |
| | Update Throttle Rate Limit | | | N | |
| | Update Throttle Command Limits | | | N | |
| | Update Mach Error Gain | | | N | |
| | Update Throttle Bandwidth | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 6 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P681 | Coordinate Vector Matrix Algebra | | | N | |
| | Matrix Operations | | | N | |
| | "+" | | | Y | MA |
| | "-" | | | Y | MA |
| | "+" | | | Y | MA |
| | "-" | | | Y | MA |
| | Set_to_Identity_Matrix | | | Y | MA |
| | Set_to_Zero_Matrix | | | Y | MA |
| | Vector Scalar Operations | 14 | 9 | N | |
| | "*" | 2 | 10 | Y | U |
| | Sparse_X_Vector_Scalar_Multiply | | | Y | MA |
| | "/" | 2 | 10 | Y | U |
| | Matrix Scalar Operations | | | N | |
| | "*" | | | Y | MA |
| | "/" | | | Y | MA |
| | Cross Product | 14 | 14 | Y | U |
| | Matrix Vector Multiply | | | Y | MA |
| | Matrix Matrix Multiply | | | Y | MA |
| | Vector Operations | 11 | 21 | N | |
| | Sparse_Right_XY_Subtract | 2 | 10 | Y | U |
| | Set_to_Zero_Vector | | | Y | MC |
| | "+" | 2 | 10 | Y | U |
| | "-" | 2 | 10 | Y | U |
| | Vector_Length | | | Y | MA |
| | Dot_Product | 2 | 10 | Y | U |
| | Sparse_Right_Z_Add | 2 | 10 | Y | U |
| | Sparse_Right_X_Add | | | Y | MA |
| | SUBTOTALS | 53 | 114 | 22 | |
| P682 | General Vector Matrix Algebra | | | N | |
| | ABA_Trans_Dynam_Sparse_Matrix_Sq_Matrix | | | N | |
| | ABA_Transpose | | | Y | MC |
| | ABA_Trans_Vector_Sq_Matrix | | | N | |
| | ABA_Transpose | | | Y | MC |
| | ABA_Trans_Vector_Scalar | | | N | |
| | ABA_Transpose | | | Y | MC |
| | Column Matrix Operations | 12 | 7 | N | |
| | Set_Diagonal_and_Subtract_from_Identity | 3 | 17 | Y | U |
| | ABA_Transpose | | | Y | MA |
| | ABA_Symm_Transpose | 9 | 37 | Y | U |
| | Dot Product Operations Unrestricted | | | N | |
| | Dot Product | | | Y | MA |
| | Dot Product Operations Restricted | | | Y | MC |
| | Diagonal Full Matrix Add Unrestricted | | | N | |
| | "+" | | | Y | DF |
| | Diagonal Full Matrix Add Restricted | 10 | 21 | Y | U |
| | Matrix Scalar Operations Constrained | | | N | |
| | "*" | | | Y | MA |
| | "/" | | | Y | MA |
| | Diagonal Matrix Scalar Operations | 15 | 11 | N | |
| | "*" | 2 | 18 | Y | U |
| | "/" | | | Y | MA |
| | Matrix Vector Multiply Unrestricted | | | N | |
| | "*" | | | Y | MC |
| | Matrix Vector Multiply Restricted | | | Y | MC |
| | Vector Matrix Multiply Unrestricted | | | N | |
| | "*" | | | Y | DF |
| | Vector Matrix Multiply Restricted | | | Y | MC |

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 7 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | Vector Vector Transpose Multiply Unrestricted | | | N | |
| | "*" | | | Y | DF |
| | Vector Vector Transpose Multiply Restricted | 16 | 16 | Y | U |
| | Matrix Matrix Multiply Unrestricted | | | N | |
| | "*" | | | Y | DF |
| | Matrix Matrix Multiply Restricted | 18 | 20 | Y | U |
| | Matrix Matrix Transpose Multiply Unrestricted | | | N | |
| | "*" | | | Y | NA |
| | Matrix Matrix Transpose Multiply Restricted | | | Y | NA |
| | Symmetric Full Storage Matrix Operations Constrained | 8 | 15 | N | |
| | Change_Element | | | Y | NC |
| | Set_to_Identity_Matrix | | | Y | NA |
| | Set_to_Zero_Matrix | 1 | 5 | Y | U |
| | Add_to_Identity | | | Y | NA |
| | Subtract_from_Identity | | | Y | NA |
| | "+" | 2 | 30 | Y | U |
| | "-" | | | Y | NA |
| | Diagonal Matrix Operations | | | N | |
| | Identity_Matrix | | | Y | NA |
| | Zero_Matrix | | | Y | NA |
| | Change_Element | | | Y | NC |
| | Retrieve_Element | | | Y | NA |
| | Row_Slice | | | Y | NA |
| | Colmn_Slice | | | Y | NA |
| | Add_to_Identity | | | Y | NA |
| | Subtract_from_Identity | | | Y | NA |
| | "+" | | | Y | NA |
| | "-" | | | Y | NA |
| | Vector Scalar Operations Unconstrained | | | N | |
| | "*" | | | Y | DF |
| | "/" | | | Y | DF |
| | Vector Scalar Operations Constrained | 14 | 5 | N | |
| | "*" | 2 | 11 | Y | U |
| | "/" | 2 | 11 | Y | U |
| | Matrix Scalar Operations Unconstrained | | | N | |
| | "*" | | | Y | NA |
| | "/" | | | Y | NA |
| | Symmetric Half Storage Matrix Operations | | | N | |
| | Initialize | | | N | |
| | Identity_Matrix | | | Y | DF |
| | Zero_Matrix | | | Y | DF |
| | Change_Element | | | Y | DF |
| | Retrieve_Element | | | Y | DF |
| | Row_Slice | | | Y | DF |
| | Column_Slice | | | Y | DF |
| | Add_to_Identity | | | Y | DF |
| | Subtract_from_Identity | | | Y | DF |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | Swap_Col | | | N | |
| | Swap_Row | | | N | |
| | Symmetric Full Storage Matrix Operations Unconstrained | | | N | |
| | Change_Element | | | Y | DF |
| | Set_to_Identity_Matrix | | | Y | DF |
| | Set_to_Zero_Matrix | | | Y | DF |
| | Add_to_Identity | | | Y | DF |
| | Subtract_from_Identity | | | Y | DF |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 8 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | Matrix Operations Unconstrained | | | N | |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | Set_to_Identity_Matrix | | | Y | DF |
| | Set_to_Zero_Matrix | | | Y | DF |
| | "*" | | | Y | DF |
| | Matrix Operations Constrained | | | N | |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | Set_to_Identity_Matrix | | | Y | DF |
| | Set_to_Zero_Matrix | | | Y | DF |
| | Dynamically_Sparse_Matrix_Operations_Unconstrained | | | N | |
| | Set_to_Identity_Matrix | | | Y | DF |
| | Set_to_Zero_Matrix | | | Y | DF |
| | Add_to_Identity | | | Y | DF |
| | Subtract_from_Identity | | | Y | DF |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | Dynamically Sparse Matrix Operations Constrained | | | N | |
| | Set_to_Zero_Matrix | | | Y | NC |
| | Add_to_Identity | | | Y | NC |
| | Subtract_from_Identity | | | Y | NC |
| | "+" | | | Y | NA |
| | "-" | | | Y | NA |
| | Set_to_Identity_Matrix | | | Y | NC |
| | Vector_Operations_Unconstrained | | | N | |
| | "+" | | | Y | DF |
| | "-" | | | Y | DF |
| | Dot_Product | | | Y | DF |
| | Vector_Length | | | Y | DF |
| | Vector Operations Constrained | 13 | 7 | N | |
| | Dot_Product | | | Y | NA |
| | Vector_Length | | | Y | NA |
| | "+" | 2 | 11 | Y | U |
| | "-" | | | Y | NA |
| SUBTOTALS | | 129 | 242 | 97 | |

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 9 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P683 | Standard Trig | 13 | | N | |
| | Arctan2 | 11 | 27 | Y | U |
| | Sin | | | Y | M |
| | Sin | | | Y | NA |
| | Sin | | | Y | NA |
| | Cos | | | Y | M |
| | Cos | | | Y | NA |
| | Cos | | | Y | NA |
| | Sin_Cos | | | Y | M |
| | Sin_Cos | | | Y | NA |
| | Sin_Cos | | | Y | NA |
| | Tan | | | Y | M |
| | Tan | | | Y | NA |
| | Tan | | | Y | NA |
| | Arcsin | | | Y | M |
| | Arcsin | | | Y | NA |
| | Arcsin | | | Y | NA |
| | Arccos | | | Y | M |
| | Arccos | | | Y | NA |
| | Arccos | | | Y | NA |
| | Arcsin_Arccos | | | Y | M |
| | Arcsin_Arccos | | | Y | NA |
| | Arcsin_Arccos | | | Y | NA |
| | Arctan | | | Y | M |
| | Arctan | | | Y | NA |
| | Arctan | | | Y | NA |
| | SUBTOTALS | 24 | 27 | 25 | |
| P684 | Geometric Operations | | | N | |
| | Unit Radial Vector | 15 | 22 | Y | U |
| | Unit Normal Vector | | | Y | NA |
| | Compute Segment and Unit Normal Vector | | | Y | DF |
| | Compute Segment and Unit Normal Vector with Arcsin | 23 | 16 | Y | U |
| | Great Circle Arc Length | | | Y | NA |
| | Compute | | | N | |
| | SUBTOTALS | 38 | 38 | 5 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 10 of 18)

| TLCSC Number | TLCSC Name Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P686 | Signal Processing | | | N | |
| | General First Order Filter | | | Y | N |
| | Update Coefficients | | | N | |
| | Filter | | | N | |
| | Reinitialize | | | N | |
| | Tustin Lead Lag Filter | | | Y | NA |
| | Update Coefficients | | | N | |
| | Filter | | | N | |
| | Reinitialize | | | N | |
| | Tustin Lag Filter | | | Y | NA |
| | Update Coefficients | | | N | |
| | Filter | | | N | |
| | Reinitialize | | | N | |
| | Second Order Filter | | | Y | NA |
| | Redefine Coefficients | | | N | |
| | Filter | | | N | |
| | Reinitialize | | | N | |
| | Tustin Integrator With Limit | | | Y | NA |
| | Update Limit | | | N | |
| | Update Gain | | | N | |
| | Integrate | | | N | |
| | Reset | | | N | |
| | Limit Flag Setting | | | N | |
| | Tustin Integrator With Asymmetric Limit | | | Y | NA |
| | Update Limits | | | N | |
| | Update Gain | | | N | |
| | Integrate | | | N | |
| | Reset | | | N | |
| | Limit Flag Setting | | | N | |
| | Upper Lower Limiter | 8 | 12 | Y | U |
| | Update Limits | 2 | 11 | N | |
| | Limit | 1 | 13 | N | |
| | Upper Limiter | 6 | 7 | Y | U |
| | Update Limit | 1 | 5 | N | |
| | Limit | 1 | 11 | N | |
| | Lower Limiter | 6 | 7 | Y | U |
| | Update Limit | 1 | 6 | N | |
| | Limit | 1 | 11 | N | |
| | Absolute Limiter | 6 | 7 | Y | U |
| | Update Limit | 1 | 5 | N | |
| | Limit | 1 | 15 | N | |
| | Absolute Limiter With Flag | | | Y | NA |
| | Limit Flag Setting | | | N | |
| | Limit | | | N | |
| | Update Limit | | | N | |
| | SUBTOTALS | 35 | 110 | 11 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 11 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P687 | General Purpose Math | | | N | |
| | Integrator | | | Y | NA |
| | Reinitialize | | | N | |
| | Update | | | N | |
| | Integrate | | | N | |
| | Interpolate or Extrapolate | | | Y | NA |
| | Square Root | 7 | | Y | N |
| | Sqrt | 1 | | N | |
| | Root Sum Of Squares | 11 | 9 | Y | U |
| | Sign | | | Y | NA |
| | Mean Value | | | Y | NA |
| | Mean Absolute Difference | | | Y | NA |
| | Two Way Table Lookup | | | Y | NA |
| | Initialize | | | N | |
| | Lookup Y | | | N | |
| | Lookup X | | | N | |
| | Lookup Table Even Spacing | | | Y | NA |
| | Initialize | | | N | |
| | Lookup | | | N | |
| | Lookup | | | N | |
| | Lookup Table Uneven Spacing | | | Y | NA |
| | Initialize | | | N | |
| | Lookup | | | N | |
| | Lookup | | | N | |
| | Incrementor | | | Y | NA |
| | Reinitialize | | | N | |
| | Increment | | | N | |
| | Decrementor | | | Y | NA |
| | Reinitialize | | | N | |
| | Decrement | | | N | |
| | Running Average | | | Y | NA |
| | Reinitialize | | | N | |
| | Reinitialize | | | N | |
| | Current Average | | | N | |
| | Accumulator | 6 | 9 | Y | U |
| | Reinitialize | 1 | 5 | N | |
| | Accumulate | 1 | 5 | N | |
| | Accumulate | 2 | 8 | N | |
| | Retrieve | 1 | 5 | N | |
| | Change Calculator | | | Y | NA |
| | Reinitialize | | | N | |
| | Change | | | N | |
| | Retrieve Value | | | N | |
| | Change Accumulator | | | Y | NA |
| | Reinitialize | | | N | |
| | Reinitialize | | | N | |
| | Accumulate Change | | | N | |
| | Accumulate Change | | | N | |
| | Retrieve Accumulation | | | N | |
| | Retrieve Previous Value | | | N | |
| | SUBTOTALS | 30 | 41 | 16 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 12 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P688 | Polynomials | | | N | |
| | Reduction Operations | 7 | 4 | N | |
| | Sine Reduction | 1 | 13 | Y | U |
| | Cosine Reduction | 1 | 6 | Y | U |
| | Taylor Series | | | N | |
| | Taylor Natural Log | | | N | |
| | Nat Log 8term | | | Y | DF |
| | Nat Log 7term | | | Y | DF |
| | Nat Log 6term | | | Y | DF |
| | Nat Log 5term | | | Y | DF |
| | Nat Log 4term | | | Y | DF |
| | Taylor Log Base N | | | N | |
| | Log Base N 8term | | | Y | NA |
| | Log N 8term | | | N | |
| | Log Base N 7term | | | Y | NA |
| | Log N 7term | | | N | |
| | Log Base N 6term | | | Y | NA |
| | Log N 6term | | | N | |
| | Log Base N 5term | | | Y | NA |
| | Log N 5term | | | N | |
| | Log Base N 4term | | | Y | NA |
| | Log N 4term | | | N | |
| | Taylor Degree Operations | | | N | |
| | Mod Cos D 4term | | | Y | DF |
| | Tan D 8term | | | Y | DF |
| | Mod Tan D 8term | | | Y | DF |
| | Mod Tan D 7term | | | Y | DF |
| | Mod Tan D 6term | | | Y | DF |
| | Mod Tan D 5term | | | Y | DF |
| | Mod Tan D 4term | | | Y | DF |
| | Sin D 8term | | | Y | DF |
| | Sin D 7term | | | Y | DF |
| | Sin D 6term | | | Y | DF |
| | Sin D 5term | | | Y | DF |
| | Mod Sin D 8term | | | Y | DF |
| | Mod Sin D 7term | | | Y | DF |
| | Mod Sin D 6term | | | Y | DF |
| | Mod Sin D 5term | | | Y | DF |
| | Mod Sin D 4term | | | Y | DF |
| | Cos D 8term | | | Y | DF |
| | Cos D 7term | | | Y | DF |
| | Cos D 6term | | | Y | DF |
| | Cos D 5term | | | Y | DF |
| | Cos D 4term | | | N | |
| | Mod Cos D 8term | | | Y | DF |
| | Mod Cos D 7term | | | Y | DF |
| | Mod Cos D 6term | | | Y | DF |
| | Mod Cos D 5term | | | Y | DF |
| | Sin D 4term | | | N | |

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 13 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | Taylor Radian Operations | | | N | |
| | Arctan R 7term | | | Y | DF |
| | Arctan R 6term | | | Y | DF |
| | Arctan R 5term | | | Y | DF |
| | Arctan R 4term | | | Y | DF |
| | Alt Arctan R 8term | | | Y | DF |
| | Alt Arctan R 7term | | | Y | DF |
| | Alt Arctan R 6term | | | Y | DF |
| | Alt Arctan R 5term | | | Y | DF |
| | Alt Arctan R 4term | | | Y | DF |
| | Mod Sin R 6term | | | Y | DF |
| | Mod Sin R 5term | | | Y | DF |
| | Mod Sin R 4term | | | Y | DF |
| | Cos R 8term | | | Y | DF |
| | Cos R 7term | | | Y | DF |
| | Cos R 6term | | | Y | DF |
| | Cos R 5term | | | Y | DF |
| | Cos R 4term | | | Y | DF |
| | Mod Cos R 8term | | | Y | DF |
| | Mod Cos R 7term | | | Y | DF |
| | Mod Cos R 6term | | | Y | DF |
| | Mod Cos R 5term | | | Y | DF |
| | Mod Cos R 4term | | | Y | DF |
| | Tan R 8term | | | Y | DF |
| | Mod Tan R 8term | | | Y | DF |
| | Mod Tan R 7term | | | Y | DF |
| | Mod Tan R 6term | | | Y | DF |
| | Mod Tan R 5term | | | Y | DF |
| | Mod Tan R 4term | | | Y | DF |
| | Arcsin R 8term | | | Y | DF |
| | Arcsin R 7term | | | Y | DF |
| | Arcsin R 6term | | | Y | DF |
| | Arcsin R 5term | | | Y | DF |
| | Arccos R 8term | | | Y | DF |
| | Arccos R 7term | | | Y | DF |
| | Arccos R 6term | | | Y | DF |
| | Arccos R 5term | | | Y | DF |
| | Arctan R 8term | | | Y | DF |
| | Sin R 8term | | | Y | DF |
| | Sin R 7term | | | Y | DF |
| | Sin R 6term | | | Y | DF |
| | Sin R 5term | | | Y | DF |
| | Sin R 4term | | | Y | DF |
| | Mod Sin R 8term | | | Y | DF |
| | Mod Sin R 7term | | | Y | DF |
| | Modified Newton Raphson | 3 | 8 | N | |
| | SqRt | 4 | 33 | Y | U |
| | Newton Raphson | | | N | |
| | SqRt | | | Y | DF |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 14 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | System Functions | | | N | |
| | Semicircle Operations | | | N | |
| | Sin | | | Y | DF |
| | Cos | | | Y | DF |
| | Tan | | | Y | DF |
| | Arcsin | | | Y | DF |
| | Arccos | | | Y | DF |
| | Arctan | | | Y | DF |
| | Degree Operations | | | N | |
| | Sin | | | Y | DF |
| | Cos | | | Y | DF |
| | Tan | | | Y | DF |
| | Arcsin | | | Y | DF |
| | Arccos | | | Y | DF |
| | Arctan | | | Y | DF |
| | Square Root | | | Y | DF |
| | Sqrt | | | N | |
| | Base 10 Logarithm | | | Y | MA |
| | Log 10 | | | N | |
| | Base N Logarithm | | | Y | MA |
| | Log N | | | N | |
| | Radian Operations | | | N | |
| | Sin | | | Y | DF |
| | Cos | | | Y | DF |
| | Tan | | | Y | DF |
| | Arcsin | | | Y | DF |
| | Arccos | | | Y | DF |
| | Arctan | | | Y | DF |
| | Cody Waite | | | N | |
| | Cody Natural Log | | | N | |
| | Nat Log | | | Y | MA |
| | R | | | N | |
| | Defloat | | | N | |
| | Cody Log Base N | | | N | |
| | Log Base N | | | Y | MA |
| | Log N | | | N | |
| | Continued Fractions | | | N | |
| | Continued Radian Operations | | | N | |
| | Tan R | | | Y | DF |
| | Arctan R | | | Y | DF |
| | Fike | 3 | 5 | N | |
| | Fike Semicircle Operations | 8 | 10 | N | |
| | Arcsin S 6term | 1 | 31 | Y | U |
| | Arccos S 6term | 1 | 32 | Y | U |
| | General Polynomial | | | N | |
| | Polynomial | | | Y | MA |
| | Hart | | | N | |
| | Hart Radian Operations | | | N | |
| | Cos R 5term | | | Y | DF |
| | Hart Degree Operations | | | N | |
| | Cos D 5term | | | Y | DF |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 15 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | Hastings | | | N | |
| | Hastings Degree Operations | | | N | |
| | Sin D 5term | | | Y | DF |
| | Sin D 4term | | | Y | DF |
| | Cos D 5term | | | Y | DF |
| | Cos D 4term | | | Y | DF |
| | Tan D 5term | | | Y | DF |
| | Tan D 4term | | | Y | DF |
| | Hastings Radian Operations | 12 | 46 | N | |
| | Cos R 5term | 1 | 16 | Y | U |
| | Cos R 4term | 1 | 15 | Y | U |
| | Tan R 5term | 1 | 12 | Y | U |
| | Tan R 4term | 1 | 12 | Y | U |
| | Arctan R 8term | 1 | 18 | Y | U |
| | Arctan R 7term | | | Y | DF |
| | Arctan R 6term | 1 | 16 | Y | U |
| | Mod Arctan R 8term | | | Y | DF |
| | Mod Arctan R 7term | | | Y | DF |
| | Mod Arctan R 6term | | | Y | DF |
| | Sin R 5term | 1 | 14 | Y | U |
| | Sin R 4term | 1 | 13 | Y | U |
| | Chebyshev | | | N | |
| | Chebyshev Radian Operations | | | N | |
| | Sin R 5term | | | Y | DF |
| | Chebyshev Degree Operations | | | N | |
| | Sin D 5term | | | Y | DF |
| | Chebyshev Semicircle Operations | | | N | |
| | Sin S 5term | | | Y | DF |
| | SUBTOTALS | 49 | 304 | 133 | |
| P691 | Abstract Data Structures | | | N | |
| | Bounded Stack | | | Y | NA |
| | Clear_Stack | | | N | |
| | Add_Element | | | N | |
| | Retrieve_Element | | | N | |
| | Peek | | | N | |
| | Stack_Status | | | N | |
| | Stack_Length | | | N | |
| | Unbounded Stack | | | Y | NA |
| | Initialize | | | N | |
| | Clear_Stack | | | N | |
| | Free_Memory | | | N | |
| | Add_Element | | | N | |
| | Retrieve_Element | | | N | |
| | Peek | | | N | |
| | Stack_Status | | | N | |
| | Stack_Length | | | N | |
| | Dot_Next | | | N | |
| | Set_Next | | | N | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 16 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| | Unbounded FIFO Buffer | | | Y | DF |
| | Initialize_Buffer | | | N | |
| | Clear_Buffer | | | N | |
| | Free_Memory | | | N | |
| | Add_Element | | | N | |
| | Retrieve_Element | | | N | |
| | Peek | | | N | |
| | Buffer_Status | | | N | |
| | Buffer_Length | | | N | |
| | Dot_Next | | | N | |
| | Set_Next | | | N | |
| | Nonblocking Circular Buffer | 20 | 9 | Y | U |
| | Clear_Buffer | 1 | 10 | N | |
| | Add_Element | 2 | 26 | N | |
| | Retrieve_Element | 2 | 19 | N | |
| | Peek | 1 | 17 | N | |
| | Buffer_Status | 1 | 14 | N | |
| | Buffer_Length | 1 | 5 | N | |
| | Unbounded Priority Queue | 28 | 52 | Y | U |
| | Queue_Status | 1 | 14 | N | |
| | Queue_Length | 1 | 9 | N | |
| | Dot_Next | 1 | 5 | N | |
| | Set_Next | 2 | 6 | N | |
| | Initialize | 1 | 16 | N | |
| | Clear_Queue | 1 | 18 | N | |
| | Free_Memory | 1 | 12 | N | |
| | Add_Element | 3 | 29 | N | |
| | Retrieve_Element | 2 | 18 | N | |
| | Peek | 1 | 12 | N | |
| | Bounded FIFO Buffer | 21 | 9 | Y | U |
| | Peek | 1 | 18 | N | |
| | Buffer_Status | 1 | 15 | N | |
| | Buffer_Length | 1 | 5 | N | |
| | Clear_Buffer | 1 | 10 | N | |
| | Add_Element | 2 | 19 | N | |
| | Retrieve_Element | 2 | 19 | N | |
| | Available Space List Operations | 0 | 6 | N | |
| | New_Node | 0 | 17 | N | |
| | Save_Node | 0 | 10 | N | |
| | Save_List | 0 | 12 | N | |
| | SUBTOTALS | 99 | 431 | 6 | |
| P692 | Abstract Processes | | | Y | M |
| | Finite State Machine | | | N | |
| | Mealy Machine | | | N | |
| | Event-Driven Sequencer | | | N | |
| | Time-Driven Sequencer | | | N | |
| | Sequence Controller | | | N | |
| | SUBTOTALS | 0 | 0 | 1 | |

# TABLE A-2. PARTS USAGE AND CODE COUNT

(Part 17 of 18)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P851 | Unit Conversions | | | N | |
| | Kilograms per Meter Squared and Pounds per Foot Squared | | | N | |
| | Conversion to Pounds per Foot2 | | | Y | NA |
| | Conversion to Kilograms per Meter2 | | | Y | NA |
| | Radians and Semicircles per Second | | | N | |
| | Conversion to Semicircles per Second | | | Y | NA |
| | Conversion to Radians per Second | | | Y | NA |
| | Degrees and Semicircles | | | N | |
| | Conversion to Semicircles | | | Y | NA |
| | Conversion to Degrees | | | Y | NA |
| | Degrees and Semicircles per Second | | | N | |
| | Conversion to Semicircles per Second | | | Y | NA |
| | Conversion to Degrees per Second | | | Y | NA |
| | Seconds and Minutes | | | N | |
| | Conversion to Minutes | | | Y | NA |
| | Conversion to Seconds | | | Y | NA |
| | Centigrade and Fahrenheit | | | N | |
| | Conversion to Fahrenheit | | | Y | NA |
| | Conversion to Centigrade | | | Y | NA |
| | Centigrade and Kelvin | | | N | |
| | Conversion to Kelvin | | | Y | NA |
| | Conversion to Centigrade | | | Y | NA |
| | Fahrenheit and Kelvin | | | N | |
| | Conversion to Kelvin | | | Y | NA |
| | Conversion to Fahrenheit | | | Y | NA |
| | Kilograms and Pounds | | | N | |
| | Conversion to Kilograms | | | Y | NA |
| | Conversion to Pounds | | | Y | NA |
| | Meters and Feet per Second | | | N | |
| | Conversion to Feet per Second | | | Y | NA |
| | Conversion to Meters per Second | 2 | 5 | Y | U |
| | Meters and Feet per Second Squared | | | N | |
| | Conversion to Feet per Second2 | | | Y | NA |
| | Conversion to Meters per Second2 | | | Y | NA |
| | Gees and Meters per Second Squared | | | N | |
| | Conversion to Meters per Second2 | 2 | 5 | Y | U |
| | Conversion to Gees | | | Y | NA |
| | Gees and Feet per Second Squared | | | N | |
| | Conversion to Feet per Second2 | | | Y | NA |
| | Conversion to Gees | | | Y | NA |
| | Radians and Degrees | | | N | |
| | Conversion to Degrees | | | Y | NA |
| | Conversion to Radians | | | Y | NA |
| | Radians and Degrees per Second | | | N | |
| | Conversion to Degrees per Second | | | Y | NA |
| | Conversion to Radians per Second | | | Y | NA |
| | Radians and Semicircles | 5 | 2 | N | |
| | Conversion to Semicircles | 1 | 5 | Y | U |
| | Conversion to Radians | 1 | 5 | Y | U |
| | Meters and Feet | | | N | |
| | Conversion to Feet | | | Y | NA |
| | Conversion to Meters | | | Y | NA |
| | SUBTOTALS | 11 | 22 | 34 | |

## TABLE A-2. PARTS USAGE AND CODE COUNT

(Concluded)

| TLCSC Number | TLCSC Name / Lower Level Units | Code Size spec | Code Size body | Part | Use Code |
|---|---|---|---|---|---|
| P852 | External Form Conversion Twos Complement | | | N | |
| | Scale | | | Y | NA |
| | Unscale | 2 | 8 | Y | U |
| | SUBTOTALS | 2 | 8 | 2 | |
| P890 | Quaternion Operations | | | N | |
| | Quaternion Computed From Euler Angles | 15 | 26 | Y | U |
| | Normalized Quaternion | | | Y | NA |
| | "*" | 4 | 24 | Y | U |
| | SUBTOTALS | 19 | 50 | 3 | |
| | TOTALS | 1,431 | 2,480 | 453 | |
| | CODE TOTALS | | 3,911 | | |

## References

[1] L.A Finch, J.F. Mason, Software Test Report for the 11th Missile Application of the Common Ada Missile Packages (CAMP) Project, AFATL-TR-88-146, Air Force Armament Laboratory, Eglin Air Force Base, Florida 32542-5434, November 1988.

INITIAL DISTRIBUTION LIST

| | |
|---|---|
| GTE GOVERNMENT SYS CORP | 1 |
| ADVANCED DIGITAL SYSTEMS | 1 |
| AFATL/FXG | 4 |
| MILITARY COMPUTER SYSTEMS | 1 |
| LOCKHEED/0/62-81, B/563, F15 | 1 |
| HUGHES/FULLERTON | 1 |
| UNISYS/MS-E1D08 | 1 |
| WESTINGHOUSE/BALTIMORE | 1 |
| AFWAL/AAAS-2 | 1 |
| BOOZ-ALLEN & HAMILTON, INC | 1 |
| BOEING AEROSPACE COMPANY/MS 8H-09 | 1 |
| BOEING AEROSPACE CO | 1 |
| AD/YGE | 1 |
| SOFTWARE PRODUCTIVITY CONSORTIUM | 5 |
| ARMY CECOM/AMSEL-COM-IA | 1 |
| NAVAL TRAINING SYS CENTER/CODE 251 | 1 |
| SCIENCE APPLICATIONS INTL CORP | 1 |
| RAYTHEON/MSL SYS DIVISION | 1 |
| CALSPAN | 1 |
| KAMAN SCIENCES CORPORATION | 1 |
| NAVAL RESEARCH LAB/CODE 5595 | 1 |
| CARNEGIE MELLON UNIV/SEI/SHOLOM | 1 |
| COLEMAN RESEARCH CORP | 1 |
| COLSA, INC | 1 |
| CONTROL DATA CORPORATION | 1 |
| WINTEC | 1 |
| CONTROL DATA/DEPT 1855 | 1 |
| DACS/RADC/COED | 1 |
| RAYTHEON/EQPT DIV | 1 |
| BMO/ACD | 1 |
| DDC-I, INC | 1 |
| ENGINEERING & ECONOMICS RESEARCH/ DIV OFFICE | 1 |
| BDM CORP | 1 |
| AFATL/FXG/EVERS | 1 |
| ESD/SYW-JPMO | 1 |
| FORD AEROSPACE & COMM CORP/MS H04 | 1 |
| UNIV OF COLORADO #202 | 1 |
| ANALYTICS | 1 |
| AFWAL/FIGL | 1 |
| WESTINGHOUSE ELECTRIC CORP/MS 5220 | 1 |
| GENERAL DYNAMICS/MZ W2-5530 | 1 |
| HONEYWELL INC | 1 |
| TAMSCO | 1 |
| STARS | 1 |
| FORD AEROSPACE/MS 2/206 | 1 |
| GRUMMAN HOUSTON CORPORATION | 1 |
| NAVAL AVIONICS CENTER/NAC-825 | 1 |
| NASA JOHNSON SPACE CENTER/EH/GHG | 1 |
| BOEING AEROSPACE/MS-8Y97 | 1 |
| HARRIS CORPORATION/GISD | 1 |
| CARNEGIE MELLON UNIV/ SOFTWARE ENGINEERING INST | 1 |
| NOAA/ERL/R/E/AL4 | 1 |
| INTERMETRICS, INC/G. RENTH | 1 |
| INTERMETRICS, INC/D.P. SMITH | 1 |
| FORD AEROSPACE/WEST DEVEL DIV | 1 |
| AD/ENE | 1 |
| ROCKWELL/MS-GA21 | 1 |
| GRUMMAN CORP/MS D-31-237 | 1 |
| INSTITUTE OF DEFENSE ANALYSIS | 1 |
| TELEDYNE BROWN/MS 178 | 1 |
| USAF/TAWC/SCAM | 1 |
| BOEING AEROSPACE CO/D. LINDBERG | 1 |
| LOGICON | 1 |
| EASTMAN KODAK/DEPT 47 | 1 |
| SYSTEMS CONTROL TECH, INC | 1 |
| E-SYSTEMS/GARLAND DIV | 1 |
| AFWAL/AAAF | 1 |
| MARTIN DEVELOPMENT | 1 |
| MA COMPUTER ASSOCIATES INC | 1 |
| IBM FEDERAL SYS DIV/MC 3206C | 1 |
| MCDONNELL DOUGLAS/INCO, INC | 1 |
| UNITED TECH, ADVANCED SYS | 1 |
| MCDONNELL AIRCRAFT CO/DEPT 300 | 1 |
| WESTINGHOUSE ELEC/MS 432 | 1 |
| MHP FU-TECH, INC | 1 |
| ITT AVIONICS | 1 |
| COSMIC/UNIV OF GA | 1 |
| NAVAL OCEAN SYS CENTER/CODE 423 | 1 |
| NAVAL WEAPONS CTR/CODE 3922 | 1 |
| ODYSSEY RESEARCH ASSOCIATES, INC | 1 |
| USA ELEC PROVING GRD/STEEP MT-DA | 1 |
| PATHFINDER SYS | 1 |
| BDM CORPORATION | 1 |
| PERCEPTRONICS, INC | 1 |
| PHOENIX INTERNATIONAL | 1 |
| MCDONNELL DOUGLAS ASTRO CO | 1 |
| GTE LABORATORY/RUBEN PRIETO-DIAZ | 1 |
| PROPRIETARY SOFTWARE SYSTEMS | 1 |
| ADVANCED TECHNOLOGY | 8 |
| STANFORD TELECOMMUNICATIONS, INC | 1 |
| RATIONAL | 1 |
| LOCKHEED MISSILES & SPACE CO | 1 |
| HERCULES DEFENSE ELEC SYS | 1 |
| AEROSPACE CORP | 1 |
| ROGERS ENGINEERING & ASSOCIATES | 1 |
| ADASOFT INC | 1 |
| ESD/XRSE | 1 |
| SANDERS/MER 24-1212 | 1 |
| CSC/ERIC SCHACHT | 1 |
| COMPUTER TECH ASSOCIATES, INC | 1 |

INITIAL DISTRIBUTION LIST (CONCLUDED)

| | |
|---|---|
| SCIENCE APPLICATIONS INTER CORP | 1 |
| HQ CASE/CBRC | 1 |
| GOULD INC/CSD | 1 |
| HQ AFSPACECOM/LKWD/STOP 32 | 1 |
| SVERDRUP/EGLIN | 1 |
| HONEYWELL INC/CLEARWATER | 1 |
| TECHNOLOGY SERVICE CORP | 1 |
| AEROSPACE/LOS ANGELES | 1 |
| SOFTWARE ARCHITECTURE & ENGIN | 1 |
| LORAL SYSTEMS GROUP/D/476-C2E | 1 |
| NADC/CODE 7033 | 1 |
| UNISYS/PAOLA RESEARCH CTR | 1 |
| SIRIUS INC | 1 |
| GENERAL RESEARCH CORP | 1 |
| SOFTECH, INC/R.L. ZALKAN | 1 |
| SOFTECH, INC/R.B. QUANRUD | 1 |
| SOFTWARE CERTIFICATION INS | 1 |
| SOFTWARE CONSULTING SPECIALIST | 1 |
| SOFTWARE PRODUCTIVITY SOLUTIONS, INC | 1 |
| STAR-GLO INDUSTRIES INC | 1 |
| NADC/CODE 50C | 1 |
| WESTINGHOUSE/BALTIMORE | 1 |
| MITRE CORPORATION | 1 |
| SYSCON CORP/I. WEBER | 1 |
| SYSCON CORP/C. MORSE | 1 |
| SYSCON CORP/T. GROBICKI | 1 |
| AEROSPACE CORPORATION/M-8-026 | 1 |
| TEXTRON DEFENSE SYSTEMS | 1 |
| GENERAL DYNAMICS/MZ 1774 | 1 |
| TIBURON SYSTEMS, INC | 1 |
| TRW DEFENSE SYS GROUP | 1 |
| NASA SPACE STATION | 1 |
| BALLISTIC MSL DEF ADVANCED/ TECHNOLOGY CENTER | 1 |
| IBM CORPORATION/FSD | 1 |
| VISTA CONTROLS CORPORATION | 1 |
| VITRO CORPORATION | 1 |
| NAVAL RESEARCH LABORATORY/CODE 5150 | 1 |
| CACI, INC | 1 |
| AFSC/PLR | 5 |
| DIRECTOR ADA JOINT PROGRAM OFFICE | 1 |
| MCDONNELL DOUGLAS ASTRONAUTICS/ E 434/106/2/MS22 | 7 |
| SDIO/S/PI | 1 |
| ADVANCED SOFTWARE TECH SPECIALTIES | 1 |
| DTIC-DDAC | 2 |
| AFCSA/SAMI | 1 |
| AUL/LSE | 1 |
| FTD/SDNF | 1 |
| AFWAL/FIES/SURVIAC | 1 |
| HQ USAFE/INATW | 1 |
| AFATL/CC | 1 |
| AFATL/CA | 1 |
| AFATL/DOIL | 2 |
| 6575 SCHOOL SQUADRON | 1 |
| IITRI | 1 |

# SUPPLEMENTARY

# INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA

AD-B129569

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

| | Technical Report Number | AD Number |
|---|---|---|
| 1. | 88-18-Vol-4 | ADB 120 251 |
| 2. | 88-18-Vol-5 | ADB 120 252 |
| 3. | 88-18-Vol-6 | ADB 120 253 |
| 4. | 88-25-Vol-1 | ADB 120 309 |
| 5. | 88-25-Vol-2 | ADB 120 310 |
| 6. | 88-62-Vol-1 | ADB 129 568 |
| 7. | 88-62-Vol-2 | ADB 129 569 |
| 8. | 88-62-Vol-3 | ADB 129-570 |
| 9. | 85-93-Vol-1 | ADB 102-654 |
| 10. | 85-93-Vol-2 | ADB 102-655 |
| 11. | 85-93-Vol-3 | ADB 102-656 |
| 12. | 88-18-Vol-1 | ADB 120 248 |
| 13. | 88-18-Vol-2 | ADB 120 249 |
| 14. | 88-18-Vol-7 | ADB 120 254 |
| 15. | 88-18-Vol-8 | ADB 120 255 |
| 16. | 88-18-Vol-9 | ADB 120 256 |
| 17. | 88-18-Vol-10 | ADB 120 257 |
| 18. | 88-18-Vol-11 | ADB 120 258 |
| 19. | 88-18-Vol-12 | ADB 120 259 |

2. If you have any questions regarding this request call me at DSN 872-4620.

Lynn S. Wargo

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO
ATTN OF: PA (Jim Swinson, 882-3931)

30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039